



TIBCO iProcess® Engine (SQL)

Administrator's Guide

Version 11.9.0

August 2022



Contents

Contents	2
The TIBCO iProcess® Engine Node	10
Table Relationships	10
nodes	11
Process Sentinels	15
Table Relationships	15
node_cluster	16
process_config	18
process_event_log	21
process_attributes	24
running_processes	27
active_logins	28
checksums	30
Mbox Sets and Message Queues	33
Table Relationships	33
iql_queues	34
mbox_set	37
mbox_set_group	39
Default SQL Database Queue Tables (Test)	41
sw_db_bgqueue_n	41
sw_db_wisqueue_n	44
sw_db_predictqueue_n	46
sw_db_deadqueue	49
Creating Additional SQL Database Queue Tables	51
Example	52

Sequence Numbers	54
About Sequence Numbers	54
Table Relationships	55
sequences	56
Procedures	58
Table Relationships	58
proc_index	59
iap_monitor	62
iap_field	64
iap_activity	65
iap_global	66
proc_version	68
procedure_lock	70
proc_instance	72
proc_audit	75
proc_defn	77
proc_deadline	80
proc_event	82
wqd_delta_subscriptions	85
Procedure Management	87
About Procedure Objects	87
Table Relationships	87
pm_objects	88
pm_objects_lock	91
pmobjects_security	93
proc_mgt_hierarchy	95
Cases	98
Table Relationships	98
case_information	99

outstanding_addr	103
wait	106
wait_step	108
status	110
case_data	112
audit_trail	115
memo	117
nmemo	120
predict	123
predict_lock	127
case_deadline_event	129
case_event	131
casenum_gaps	134
Work Items	137
Table Relationships	137
staffo	137
pack_data	142
pack_memo	145
pack_nmemo	148
qaccess	150
Case Data Queue Parameters	153
Table Relationships	153
cdqp_def	153
cdqp_cfg	155
Queue Participation and Redirection	158
Table Relationships	158
part_defn	158
part_list	160
redir_defn	162

Administrative Tables	165
Table Relationships	165
flag_table	165
version	167
Users and Work Queues	170
About User Tables	170
Table Relationships	170
user_names	171
user_attrib	173
user_setting	176
user_values	177
user_memb	180
leavers	182
tsys_user_names	184
tsys_user_attrib	185
tsys_user_values	185
tsys_user_memb	186
Roles	187
About Roles	187
Table Relationships	187
role_users	188
tsys_role_users	190
TIBCO iProcess Tables	191
About TIBCO iProcess Tables	191
Table Relationships	192
db_names	193
db_fields	195
db_values	197
tsys_db_names	199

tsys_dbs_fields	200
tsys_dbs_values	200
str_dbs_names	201
str_dbs_fields	201
ttmp_dbs_names	201
ttmp_dbs_fields	202
ttmp_dbs_values	202
Lists	203
About Lists	203
Table Relationships	204
list_names	204
list_values	206
tsys_list_names	208
tsys_list_values	209
ttmp_list_names	209
ttmp_list_values	210
iProcess Server Plug-ins	211
Table Relationships	211
eai_registry	211
Firewall Port Ranges	214
Table Relationships	214
port_range	215
port_range_active	217
port_range_conf	219
port_range_nodes	221
WQS/WIS Shared Memory	224
Table Relationships	224
wqs_index	224

System Event Logging	228
Table Relationships	228
system_event	228
system_event_conf	230
Monitoring	233
Table Relationships	233
stat_gen_mins	234
stat_gen_hours	237
stat_gen_daily	240
stat_mq_mins	244
stat_mq_hours	246
stat_mq_daily	248
stat_rp_mins	250
stat_rp_hours	253
stat_rp_daily	256
stat_vers_daily	258
stat_proc_daily	260
Views	263
SSOLite Stored Procedures	264
Overview	264
Using SSOLite Stored Procedures	265
Processing Asynchronous Message	265
Transactional Processing	265
Handling Exceptions	266
Processing Queues	269
Prioritizing Messages	270
Data Procedures	271
SW_ADD_PACK_DATA	271
SW_ADD_PACK_MEMO	273

SW_CLEAR_PACK_CACHE	274
SW_MODIFY_CASEDATA	275
Command Procedures	277
SW_AUDIT	277
SW_CASEREOPEN	279
SW_CASESTART	280
SW_CLOSE	282
SW_CLOSE_WITHOUT_EVENT	284
SW_DELAYED_RELEASE_ERR	285
SW_EVENT	286
SW_EVENT_UPDATE_PACK	288
SW_GETCASE_STATUS	289
SW_GETCASE_STATUS_EX	291
SW_GRAFT	292
SW_GRAFTCOUNT	294
SW_JUMPTO	296
SW_JUMPTO_MULTI	297
SW_PURGE	299
SW_PURGE_WITHOUT_EVENT	301
SW_SUSPEND	302
SW_ACTIVATE	303
Control Procedures	305
SW_ENABLECACHING	305
SW_DISABLECACHING	306
SW_SET_MBOX	307
SW_SET_PRIORITY	308
SW_SET_QUEUE	309
SW_UNSET_MBOX	310
SW_UNSET_PRIORITY	311
SW_UNSET_QUEUE	311
SW_INIT_TABLES	311
Debug Procedures	312

SW_SET_DEBUG	313
SW_GET_DEBUG	313
SW_CLEAR_DEBUG	314
Database Stored Procedures	315
Overview	315
CASENUM_FIND_GAPS	315
XPC_CREATE_INDEX	318
Unused Tables	320
TIBCO Documentation and Support Services	321
Legal and Third-Party Notices	323

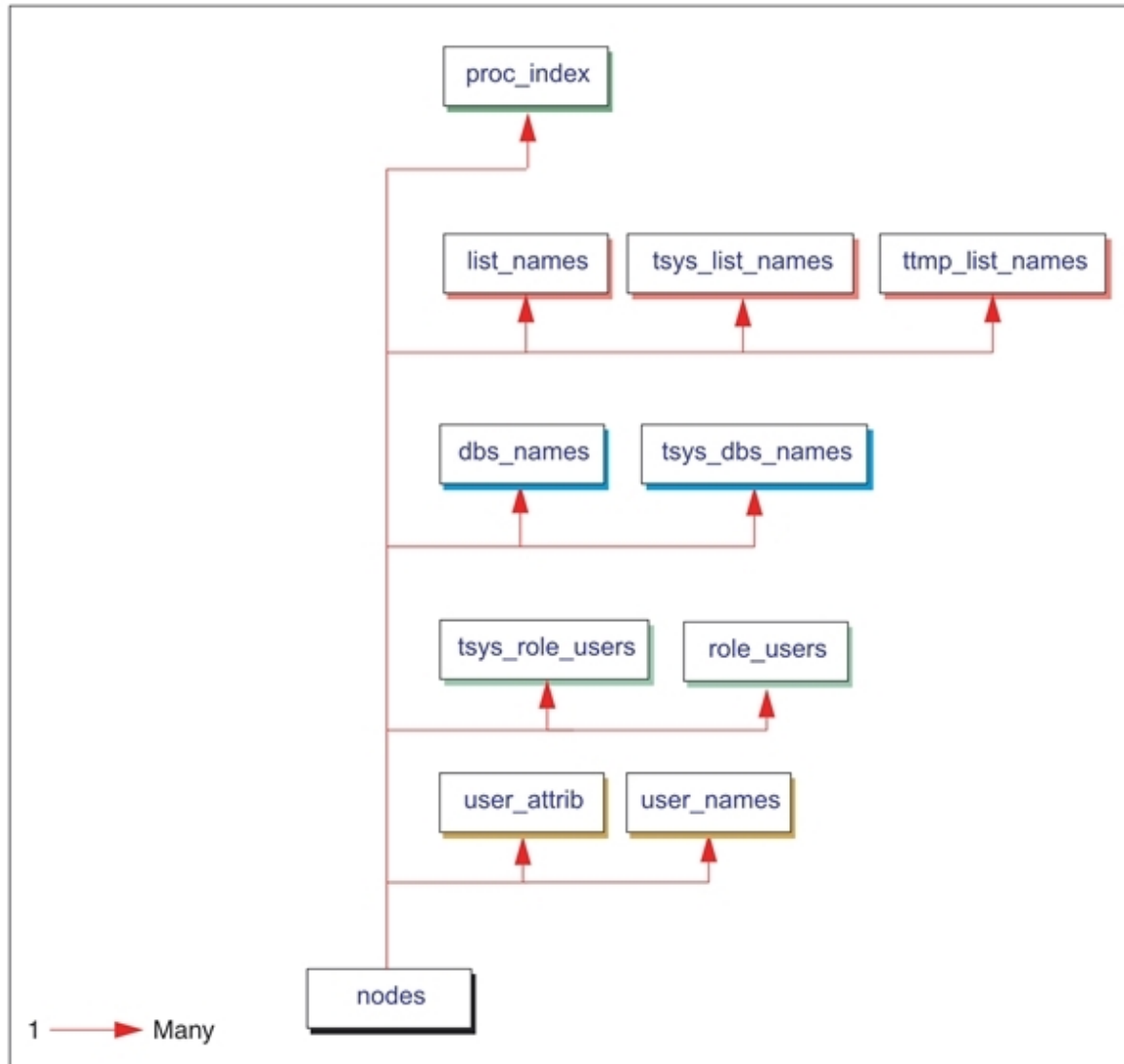
The TIBCO iProcess® Engine Node

This section describes the table that is used to store information about the TIBCO iProcess® Engine node.

Table Relationships

The following diagram shows how the [nodes](#) table is related to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



nodes

The nodes table holds information about this iProcess Engine node. A node is a single logical iProcess Engine, which may be installed either on a single computer, or spread over several using a node cluster architecture).

Structure

The nodes table has the following structure:

TABLE nodes(

node_id	INTEGER	NOT NULL,
node_name	VARCHAR(24)	NOT NULL,
dir_name	VARCHAR(28)	NOT NULL,
mail_addr	VARCHAR(149)	NULL,
mail_cert	VARCHAR(31)	NULL,
mail_type	INTEGER	NOT NULL,
node_public	SMALLINT	NOT NULL,
node_slave	SMALLINT	NOT NULL,
node_deleted	SMALLINT	NOT NULL,
rpc_majvers	SMALLINT	NOT NULL,
rpc_minvers	SMALLINT	NOT NULL,
server_majvers	SMALLINT	NOT NULL,
server_minvers	SMALLINT	NOT NULL)

Column	Description
node_id	Unique ID of this iProcess node. Note: This value is always 1.
node_name	Logical name for this node.
dir_name	Name of the directory which holds the node's data (<i>SWDIR</i>).

Column	Description
mail_addr	<i>Not used. Reserved for possible future use.</i>
mail_cert	<i>Not used. Reserved for possible future use.</i>
mail_type	<i>Not used. Reserved for possible future use.</i>
node_public	<i>Not used. Reserved for possible future use.</i>
node_slave	<i>Not used. Reserved for possible future use.</i>
node_deleted	<i>Not used. Reserved for possible future use.</i>
rpc_majvers	<i>Not used. Reserved for possible future use.</i>
rpc_minvers	<i>Not used. Reserved for possible future use.</i>
server_majvers	<i>Not used. Reserved for possible future use.</i>
server_minvers	<i>Not used. Reserved for possible future use.</i>

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_nodes	node_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_nodes	DELETE	proc_index user_names user_attrib dbs_names tsys_dbs_names list_names tsys_list_names ttmp_list_names role_users tsys_role_users

Indexes

None.

Table Activity

The nodes table contains one row, which is the entry for the iProcess Engine.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	never.
updated	never.
deleted	never.

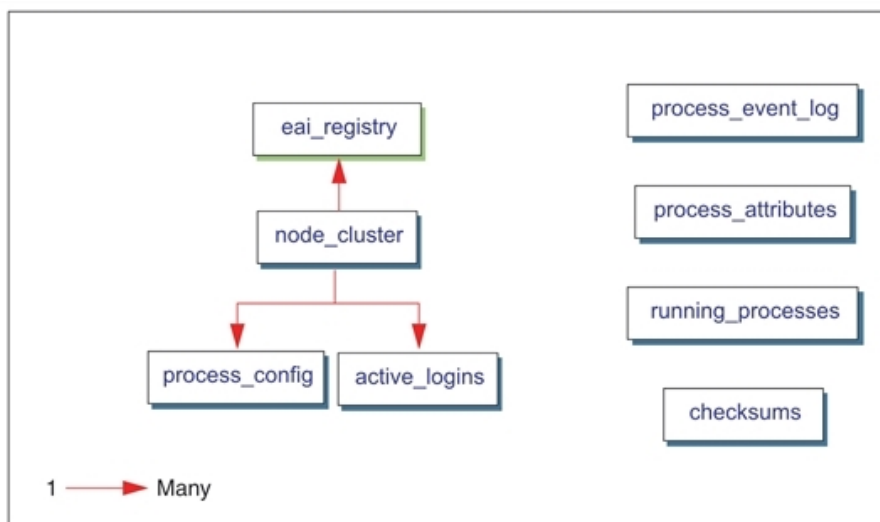
Process Sentinels

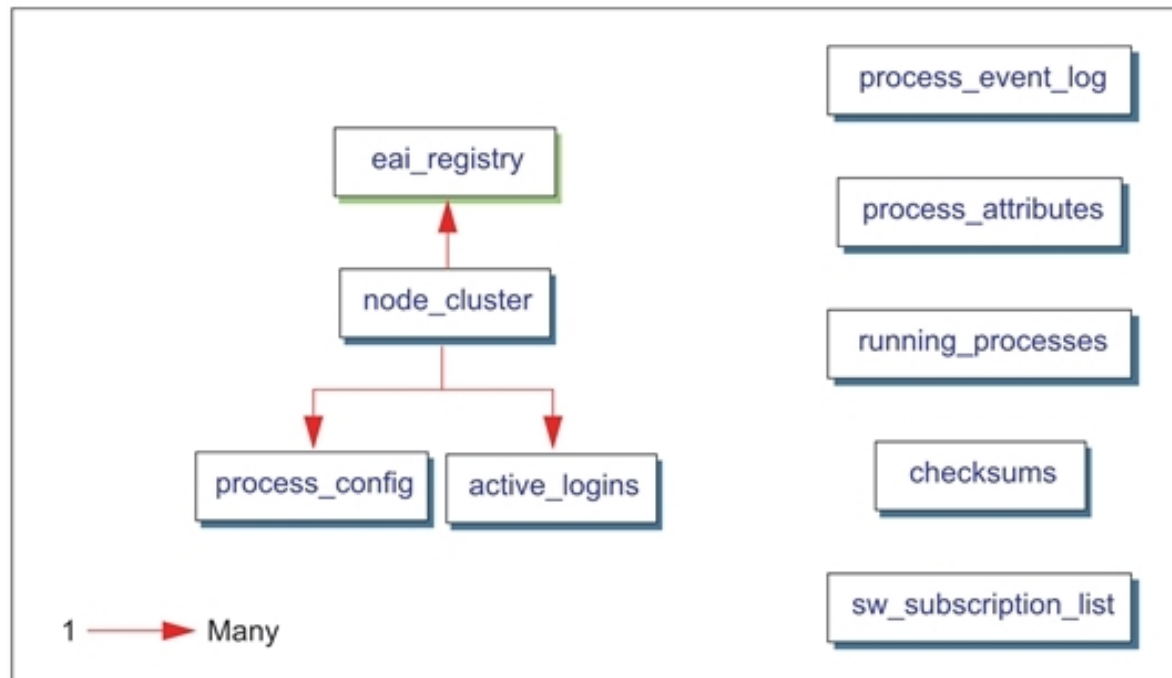
This section describes the tables that are used to store information used by the Process Sentinels.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.





node_cluster

The `node_cluster` table defines the server computers that make up this iProcess Engine node.

Structure

The `node_cluster` table has the following structure:

```
TABLE node_cluster(
```

<code>logical_machine_id</code>	INTEGER	NOT NULL,
<code>physical_machine_name</code>	VARCHAR(256)	NOT NULL,
<code>master</code>	SMALLINT	NOT NULL,

check_error_files	SMALLINT	NOT NULL,
machine_comment	VARCHAR(256)	NULL)

Column	Description
logical_machine_id	Unique ID for this server.
physical_machine_name	If a UNIX server, the name of this server (as returned by the UNIX uname command). If a Windows server, then the name of this server or the Microsoft Windows cluster network name.
master	Flag that defines whether this computer is acting as the master server (1) or, if a node-cluster architecture is being used, as a slave server (0).
check_error_files	Flag that defines whether the Process Sentinels on this server check (1) or do not check (0) for the creation of <i>SWDIR\logs\sw_error</i> and <i>sw_warn</i> files.
machine_comment	Descriptive comment describing this server.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_node_cluster	logical_machine_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_node_cluster	DELETE	eai_registry process_config active_logins

Indexes

None.

Table Activity

The `node_cluster` table contains one row for each server computer that is part of the iProcess Engine node.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new server is added to the node, either at installation or by using the <code>SWDIR\util\swadm</code> utility.
updated	a server's details are updated, using the <code>SWDIR\util\swadm</code> utility.
deleted	a server is removed from the node, using the <code>SWDIR\util\swadm</code> utility.

process_config

The `process_config` table stores information about each process *instance* that is defined on the system.

Multiple instances of each server process can be used to optimize iProcess Engine efficiency - for example, to increase the processing capability on one server, or to spread the processing load across multiple servers.

Structure

The process_config table has the following structure:

```
TABLE process_config(
  logical_machine_id          INTEGER          NOT NULL,
  logical_process_name        VARCHAR(10)      NOT NULL,
  logical_process_instance    INTEGER          NOT NULL,
  enabled                     SMALLINT         NOT NULL,
  persistent                  SMALLINT         NOT NULL,
  last_known_status           VARCHAR(20)      NOT NULL,
  status_comment              VARCHAR(255)     NULL)
```

Column	Description
logical_machine_id	ID of the server where this process instance runs, as defined in the node_cluster table.
logical_process_name	Logical name of this process instance. Note: For a list of logical process names, see the "Administering iProcess Engine Server Processes" topic in <i>TIBCO iProcess Engine Administrator's Guide</i> .
logical_process_instance	Unique ID for this process instance.
enabled	Flag that defines whether this process instance starts automatically (1) when the iProcess Engine starts, or whether it must be started manually (0).
persistent	Flag that defines whether this process instance is automatically restarted (1)

Column	Description
	<p>or not (0) when the iProcess Engine is shut down and restarted.</p> <p>Note: Any row in which the persistent value is 0 is deleted when the iProcess Engine starts up.</p>
last_known_status	<p>Last known status of this process instance, as reported to the Process Sentinels by the process.</p> <p>Either: STARTING, RUNNING, PAUSED, SUSPENDED, SHUTTING DOWN or STOPPED.</p> <p>Note: The process_event_log table provides an audit trail of changes to the status of a process instance.</p>
status_comment	Brief explanation of the last_known_status, as reported to the Process Sentinels by the process.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_process_config	logical_machine_id logical_process_name logical_process_instance

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_process_config_fk	logical_machine_id

Table Activity

The process_config table contains one row for each instance of each server process defined on the system.

Rows are added, updated, and deleted in the following situations.

A row is...	When...
added	a new process instance is added, either at installation or by using the <i>SWDIR\util\swadm</i> , <i>SWDIR\util\swsvrmgr</i> utilities. or the iProcess Server Manager.
updated	a process instance's settings or status are updated, either by system activity, or by using the <i>SWDIR\util\swadm</i> or <i>SWDIR\util\swsvrmgr</i> utilities. or the iProcess Server Manager.
deleted	a process instance is deleted, either at installation or by using the <i>SWDIR\util\swadm</i> or <i>SWDIR\util\swsvrmgr</i> utilities. or the iProcess Server Manager.

process_event_log

The process_event_log table logs all changes in the status of server process instances.

Structure

The process_event_log table has the following structure:

```
TABLE process_event_log
```

logical_machine_id	INTEGER	NOT NULL,
logical_process_name	VARCHAR(10)	NOT NULL,
logical_process_instance	INTEGER	NOT NULL,
process_id	INTEGER	NOT NULL,
process_status	INTEGER	NOT NULL,
process_status_comment	VARCHAR(255)	NULL,
timestamp	DATETIME	NOT NULL)

Column	Description
logical_machine_id	ID of the server where the process instance that this event applies to is running, as defined in the node_cluster table.
logical_process_name	Logical name of the process that this event applies to. Note: See "Administering iProcess Engine Server Process" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of logical process names.
logical_process_instance	ID of the process instance that this event applies to, as defined in the process_config table.
process_id	Process ID (PID) of the process instance that this event applies to.
process_status	Status change event that occurred for the specified process instance. One of the following: <ul style="list-style-type: none"> 3000 - process instance started. 3001 - process instance stopping.

Column	Description
	<ul style="list-style-type: none"> • 3002 - process instance stopped. • 3003 - process instance died. • 3004 - process instance paused. • 3005 - process instance unpaused.
process_status_comment	Description of the process_status entry, as reported to the Process Sentinels by the process.
timestamp	Date and time that this event occurred.

Primary Key

None.

Triggers

None.

Indexes

None.

Table Activity

The process_event_log table contains one row for each status change event that has occurred to each instance of a server process.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a process starts, receives a shutdown command, or shuts down.

A row is...	When...
updated	never.
deleted	never.

Note: Because rows are never deleted automatically, TIBCO recommend that you regularly monitor the size of this table and delete or archive rows manually if you need to.

process_attributes

The process_attributes table stores process attribute definitions, which provide configuration information for iProcess Engine server processes.

Structure

The process_attributes table is structured as follows:

TABLE process_attributes (
logical_machine_id	INTEGER	NOT NULL,
logical_process_name	VARCHAR(10)	NOT NULL,
logical_process_instance	INTEGER	NOT NULL,
attribute_name	VARCHAR(50)	NOT NULL,
attribute_value	VARCHAR(1024)	NOT NULL,
attribute_type	VARCHAR(2)	NOT NULL)

Column	Description
logical_	ID of the server where the process instance that this attribute applies to is

Column	Description
machine_id	<p>running, as defined in the node_cluster table.</p> <p>A value of 0 means that this attribute applies to all servers that are part of this node.</p>
logical_process_name	<p>Logical name of the process that this attribute applies to.</p> <p>A value of ALL means that this attribute applies to all processes on the indicated server.</p> <p>Note: See "Administering iProcess Engine Server Processes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of logical process names.</p>
logical_process_instance	<p>ID of the process instance that this attribute applies to, as defined in the process_config table.</p> <p>A value of 0 means that this attribute applies to all instances of the indicated process.</p>
attribute_name	<p>Name of this process attribute.</p> <p>Note: See "Administering Process Attributes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of the available process attributes.</p>
attribute_values	<p>Value of this process attribute.</p> <p>Note: See "Administering Process Attributes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of the valid values for each process attributes.</p>
attribute_type	<p>Type of this process attribute: either I (Integer), C (Character) or S (String).</p> <p>Note: All attribute_values are stored as strings in this table. This value determines how the value is returned to the <i>SWDIR\bin\swadm</i> interface.</p>

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_process_attributes	logical_machine_id logical_process_name logical_process_instance attribute_name

Triggers

None.


Indexes

None.

Table Activity

The process_attribute table contains one row for each unique definition of a process attribute on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new process attribute definition is added, either at installation or by using the <i>SWDIR\util\swadm</i> utility.
updated	a process attribute definition is updated, using the <i>SWDIR\util\swadm</i> utility.
deleted	a process attribute definition is deleted, using the <i>SWDIR\util\swadm</i> utility.
 Note	This table can contain orphan rows in which data can exist that does not apply to any process currently being used.

running_processes

The `running_processes` table stores information about each process instance that is currently running on the system.

Structure

The `running_processes` table has the following structure:

```
TABLE running_processes (
    logical_machine_id    INTEGER          NOT NULL,
    logical_process_name  VARCHAR(10)      NOT NULL,
    logical_process_instance INTEGER        NOT NULL,
    process_id            INTEGER          NOT NULL,
    port_number           INTEGER          NOT NULL)
```

Column	Description
<code>logical_machine_id</code>	ID of the server where this process instance is running, as defined in the node_cluster table.
<code>logical_process_name</code>	Logical name of this process instance. Note: See "Administering iProcess Engine Server Processes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of logical process names.
<code>logical_process_instance</code>	ID of this process instance, as defined in the process_config table.
<code>process_id</code>	Process ID (PID) of this process instance.
<code>port_number</code>	Port number that this process instance is running on.

Primary Key

None.

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_running_processes_fk	logical_machine_id logical_process_name logical_process_instance

Table Activity

The `running_processes` table contains one row for each instance of an iProcess Engine server process that is currently running on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new process instance is started.
updated	a process instance is restarted (<code>process_id</code> and <code>port_number</code> are updated).
deleted	a process instance is stopped.

active_logins

The `active_logins` table stores details of all users who are currently logged in to this iProcess Engine node.

Structure

The `active_logins` table has the following structure:

```
TABLE active_logins(
    logical_machine_id    INTEGER          NOT NULL,
    logical_process_name  VARCHAR(10)      NOT NULL,
    logical_process_instance INTEGER        NOT NULL,
    user_name             VARCHAR(64)       NOT NULL,
    user_id               VARCHAR(37)       NOT NULL,
    process_id            INTEGER          NOT NULL,
    filsh                 INTEGER          NOT NULL,
    windows               SMALLINT         NOT NULL,
    station_id            VARCHAR(32)       NOT NULL)
```

Column	Description
logical_machine_id	ID of the server where the process that made the login request is running, as defined in the node_cluster table.
logical_process_name	Logical name of this process instance. Note: See "Administering iProcess Engine Server Processes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of logical process names.
logical_process_instance	ID of this process instance, as defined in the process_config table.
user_name	Name of the user who is logged in, as defined in the user_names table.
user_id	ID of the user who made the login request (for internal use only).
process_id	Process ID (PID) of the process that made the login request.
filsh	FIL session handle (for internal use only).
windows	Flag that defines whether the login request came from TIBCO iProcess Objects (0) or from an TIBCO iProcess® Workspace or other SAL application (1).
station_id	Comment that identifies where a user is logged in.

Primary Key

None.

Triggers

None.

Indexes

The following indexes are defined for this table.

Index Name	Column(s) Indexed
idx_active_logins_fk	logical_machine_id
idx_active_logins ¹	user_id

Table Activity

The active_logins table contains one row for each user who is currently logged into this iProcess Engine node.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a user is logged in.
updated	never.
deleted	a user is logged out or the iProcess Engine shuts down.

checksums

The checksums table is used internally by the iProcess Engine to provide security checks on the [active_logins](#) and [port_range](#) tables.

¹This is a clustered index.

Structure

The checksums table has the following structure:

```
TABLE checksums (
```

area_id	INTEGER	NOT NULL,
area_name	VARCHAR(20)	NOT NULL,
check_sum	VARCHAR(54)	NOT NULL)

Column	Description
area_id	Unique ID of the area using this checksum
area_name	Name of the area using this checksum. Currently this is always PORT RANGING.
check_sum	Encrypted checksum for the indicated area.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_checksums	area_id

Triggers

None.

Indexes

None.

Table Activity

The checksums table contains one row for each checksum used internally by the iProcess Engine.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	the iProcess Engine is started.
updated	a login is performed.
deleted	never.

Mbox Sets and Message Queues

This section describes the tables that are used to control the behavior of the message queuing system.

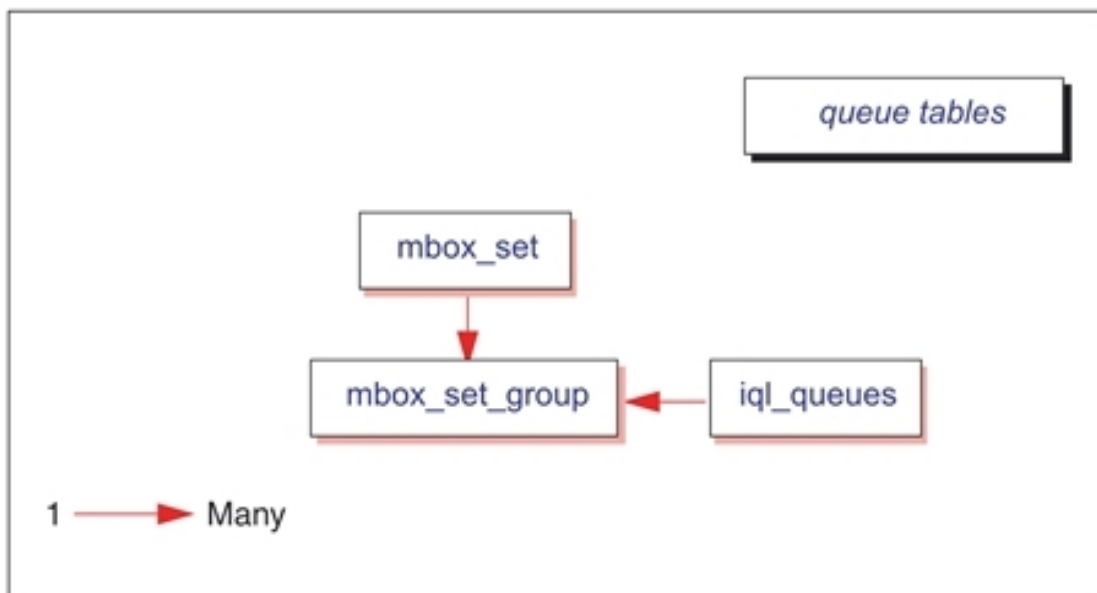
It also describes the [Default SQL Database Queue Tables \(Test\)](#), which provide the underlying message queuing system used by the iProcess message queues.

- [Default SQL Database Queue Tables \(Test\)](#)
- [Creating Additional SQL Database Queue Tables](#)

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



iqL_queues

The `iqL_queues` table defines each message queue that is available on this iProcess Engine node.

Structure

The `iqL_queues` table has the following structure:

```
TABLE iqL_tables (
  queue_id          INTEGER          NOT NULL,
  queue_name        VARCHAR(24)      NOT NULL,
  queue_type        SMALLINT         NOT NULL,
  queue_phys_descr  VARCHAR(100)     NOT NULL)
```

Column	Description
<code>queue_id</code>	Unique identifier for this message queue.
<code>queue_name</code>	Name of this message queue.
<code>queue_type</code>	Message type used by this message queue. This value is always 1, for local messages.
<code>queue_phys_descr</code>	ID of the SQL database queue table that is used to hold this message queue. See: Format of the SQL Database Queue Table ID for a description of the format used for this value. Default SQL Database Queue Tables (Test) for more information about the default SQL database queue tables, and how to create additional tables.

Format of the SQL Database Queue Table ID

The ID of the SQL database table that is used to hold this message queue (in the `queue_phys_descr` column) is specified using the following format:

```
0003: [database_name.] [owner.] queue_table
```

where:

- 0003 indicates that the remainder of the string uses SQL Server format.
- *database_name* is the name of the database that holds this *queue_table*. If this option is omitted, the iProcess database is used by default.



If you specify a different database, it *must* reside on the same SQL Server as the iProcess database.

- *owner* is the username of the user that owns this *queue_table*. If this option is omitted, the iProcess background user owns the table by default.
- *queue_table* is the name of the SQL database table used to hold this message queue. Each individual queue must be held in its own database table.

For example, the entry:

```
0003:sw_db_bgqueue_1
```

describes the SQL database table called `sw_db_bgqueue_1`, which is stored in the default iProcess database and owned by the iProcess background user.

The entry:

```
0003:sw.swpro1.sw_db_bgqueue_3
```

describes the SQL database table called `sw_db_bgqueue_3`, which is stored in the `sw` database (on the SQL Server hosting the iProcess database) and owned by user `swpro1`.

Default Message Queues and SQL Database Queue Tables

When the iProcess Engine is installed, the `init2Ksql.sql` script creates the following default set of message queues and SQL database queue tables required by the system.

Queue Name	SQL Database Table Queue ID
BGMBX1	0003:swpro.sw_db_bgqueue_1
BGMBX2	0003:swpro.sw_db_bgqueue_2
WISMBX1	0003:swpro.sw_db_wisqueue_1
WISMBX2	0003:swpro.sw_db_wisqueue_2

Queue Name	SQL Database Table Queue ID
DEADQUEUE	0003:swpro.sw_db_deadqueue
PREDICTMBOX1	0003:swpro.sw_db_predictqueue_1
PREDICTMBOX2	0003:swpro.sw_db_predictqueue_2

See [Default SQL Database Queue Tables \(Test\)](#) for a detailed description of these tables.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_iql_queues	queue_id

Triggers

None.

Indexes

None.

Table Activity

The iql_queues table contains one row for each message queue that is available on this node.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new message queue is added to the node, either at installation or by using the <i>SWDIR\util\swadm</i> utility.

A row is...	When...
updated	a message queue's details are updated, using the <i>SWDIR\util\swadm</i> utility.
deleted	a message queue is deleted from the node, using the <i>SWDIR\util\swadm</i> utility.

mbox_set

The `mbox_set` table defines the list of Mbox sets that are available on this iProcess Engine node.

Structure

The `mbox_set` table has the following structure:

```
TABLE mbox_set (
    mbox_set_id      INTEGER          NOT NULL,
    mbox_set_name    VARCHAR(32)     NOT NULL,
    mbox_set_msgtype SMALLINT        NOT NULL)
```

Each row provides the following information about a Mbox set.

Column	Description
<code>mbox_set_id</code>	Unique identifier for this Mbox set.
<code>mbox_set_name</code>	Name of this Mbox set.
<code>mbox_set_msgtype</code>	Message type used by this Mbox set. This value is always 1, for local messages.

Default Mbox Sets and Message Queues

When the iProcess Engine is installed, the `init2ksql.sql` script creates the following default Mbox sets that are required by the system. (The [mbox_set_group](#) table defines which message queues are stored in which Mbox set.)

Mbox Set	Contains these message queues
BGMBSET	BGMBBOX1, BGMBBOX2
WMDMBSET	WISMBBOX1, WISMBBOX2
PREDICTMBSET	PREDICTMBOX1, PREDICTMBOX2

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_mbox_set	mbox_set_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_mbox_set	DELETE	mbox_set_group

Indexes

None.

Table Activity

The `mbox_set` table contains one row for each Mbox set that is available on this iProcess Engine node.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new Mbox set is added to the node, either at installation or by using the <i>SWDIR\util\swadm</i> utility.
updated	an Mbox set's details are updated, using the <i>SWDIR\util\swadm</i> utility.
deleted	an Mbox set is deleted from the node, using the <i>SWDIR\util\swadm</i> utility.

mbox_set_group

The `mbox_set_group` table defines the list of individual message queues that are stored in each Mbox set.

Structure

The `mbox_set_group` table has the following structure:

```
TABLE mbox_set_group (
    mbox_set_id    INTEGER    NOT NULL,
    mbox_queue_id  INTEGER    NOT NULL)
```

Column	Description
<code>mbox_set_id</code>	Unique identifier of the Mbox set that contains the associated message queue, as defined in the mbox_set table.
<code>mbox_queue_id</code>	Unique identifier of the message queue that is included in the associated Mbox set, as defined in the iql_queues table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_mbox_set_group	mbox_set_id mbox_queue_id

Triggers

None.

Indexes

The following indexes are defined for this table.

Index Name	Indexed Column(s)
idx_mbox_set_id_fk	mbox_set_id
idx_mbox_queue_id_fk	mbox_queue_id

Table Activity

The `mbox_set_group` table contains one row for each message queue that is available on this node.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new message queue is added to the node, either at installation or by using the <code>SWDIR\util\swadm</code> utility.
updated	never.
deleted	a message queue is deleted from the node, using the <code>SWDIR\util\swadm</code> utility.

Default SQL Database Queue Tables (Test)

Each message queue defined in the [iql_queues](#) table must be mapped to its own SQL database queue table.

When the iProcess Engine is installed, the `init2ksql.sql` script creates the default set of queue tables required by the system (see [Default Message Queues and SQL Database Queue Tables](#)).

This section describes the format of each of the default queue tables.

Queue Table	See
sw_db_bgqueue_1	sw_db_bgqueue_n
sw_db_bgqueue_2	
sw_db_wisqueue_1	sw_db_wisqueue_n
sw_db_wisqueue_2	
sw_db_predictqueue_1	sw_db_predictqueue_n
sw_db_predictqueue_2	
sw_db_deadqueue	sw_db_deadqueue

If you subsequently decide to add additional message queues to your system, you must manually create the queue tables needed by those message queues. See [Creating Additional SQL Database Queue Tables](#) for more information about how to do this.

sw_db_bgqueue_n

Each `sw_db_bgqueue_n` (where *n* is 1 or 2) queue table holds messages intended for the background processes:

- iProcess processes (for example, WIS, DLMGR or RPC_POOL) enqueue messages to the table.
- The background processes (BG) dequeue and process messages from the table.

Structure

The `sw_db_bgqueue_n` table has the following structure:

```
TABLE sw_db_bgqueue_n (
    rowid          NUMERIC(15)          identity(1,1),
    last_failed    NUMERIC(10)          NULL,
    failure_count   INTEGER             NOT NULL,
    msg_id         uniqueidentifier     NOT NULL,
    msg_hdr        VARCHAR(512)         NULL,
    msg_data       VARCHAR(1024)        NOT NULL
    priority       INTEGER             NOT NULL)
```

Column	Description
<code>rowid</code>	Identifier of the row in the table for this message.
<code>last_failed</code>	<p>Number of seconds since January 1st, 1970, when this message last failed to be processed.</p> <p>When this value equals or exceeds the value of the <code>IQL_RETRY_DELAY</code> process attribute, the message is retried.</p>
<code>failure_count</code>	<p>Number of times that this message has failed to be processed.</p> <p>When this value equals or exceeds the value of the <code>IQL_RETRY_COUNT</code> process attribute, the message is moved to the sw_db_deadqueue.</p>
<code>msg_id</code>	Unique identifier of this message.
<code>msg_hdr</code>	Header data associated with this message.
<code>msg_data</code>	Message data.
<code>priority</code>	<p>Message queue priority. The lower this value is, the higher the message queue priority is.</p> <p>The default value is 50.</p>

**Note**

See "Administering Process Attributes" in *TIBCO iProcess Engine Administrator's Guide* for more information about the IQL_RETRY_DELAY and IQL_RETRY_COUNT attributes.

Primary Key

No primary key is defined for this table.

Triggers

None.

Indexes

The following clustered index is defined for this table.

Index Name	Indexed Column(s)
idx_sw_db_bgqueue_n ¹	row_id

Table Activity

Each sw_db_bgqueue_n table contains one row for each enqueued message. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	an iProcess process enqueues a message to this table.
updated	a BG process dequeues a message from this table but cannot successfully process it (but the message has not exceeded the IQL_RETRY_COUNT value).
deleted	a BG process dequeues a message from this table and either:

¹

A row is...	When...
-------------	---------

- | | |
|--|--|
| | <ul style="list-style-type: none"> successfully processes it. cannot successfully process it, and moves it to the sw_db_deadqueue table because it has exceeded the IQL_RETRY_COUNT value. |
|--|--|

sw_db_wisqueue_n

Each `sw_db_wisqueue_n` (where n is 1 or 2) queue table holds messages intended for the Work Item Server Mbox daemon process:


- The background processes (BG) enqueue messages to the table.
- The Work Item Server Mbox daemon process (WISMBD) dequeues and processes messages from the table, which it then forwards on to the Work Item Server (WIS) processes.

Structure

The `sw_db_wisqueue_n` table has the following structure:

```
TABLE sw_db_wisqueue_n (
  rowid          NUMERIC(15)          identity(1,1),
  last_failed    NUMERIC(10)          NULL,
  failure_count  INTEGER              NOT NULL,
  msg_id         uniqueidentifier     NOT NULL,
  msg_hdr        VARCHAR(512)         NULL,
  msg_data       VARCHAR(1024)        NOT NULL
  priority       INTEGER              NOT NULL)
```

Column	Description
rowid	Identifier of the row in the table for this message.
last_failed	Number of seconds since January 1st, 1970, when this message last failed to be processed.

Column	Description
	When this value equals or exceeds the value of the IQL_RETRY_DELAY process attribute, the message is retried.
failure_count	<p>Number of times that this message has failed to be processed.</p> <p>When this value equals or exceeds the value of the IQL_RETRY_COUNT process attribute, the message is moved to the sw_db_deadqueue.</p>
msg_id	Unique identifier of this message.
msg_hdr	Header data associated with this message.
msg_data	Message data.
priority	<p>Message queue priority. The lower this value is, the higher the message queue priority is.</p> <p>The default value is 50.</p>
 Note	See "Administering Process Attributes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for more information about the IQL_RETRY_DELAY and IQL_RETRY_COUNT attributes.

Primary Key

No primary key is defined for this table.

Triggers

None.

Indexes

The following clustered index is defined for this table.

Index Name	Indexed Column(s)
idx_sw_db_wisqueue_n ¹	row_id

Table Activity

Each `sw_db_wisqueue_n` table contains one row for each enqueued message. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a BG process enqueues a message to this table.
updated	the WISMBD process dequeues a message from this table but cannot successfully process it (but the message has not exceeded the <code>IQL_RETRY_COUNT</code> value).
deleted	the WISMBD process dequeues a message from this table and either: <ul style="list-style-type: none"> successfully processes it. cannot successfully process it, and moves it to the sw_db_deadqueue table because it has exceeded the <code>IQL_RETRY_COUNT</code> value.

sw_db_predictqueue_n

Each `sw_db_predictqueue_n` (where *n* is 1 or 2) queue table holds messages intended for the background case prediction server processes:

- iProcess processes (for example, WIS, DLMGR or RPC_POOL) enqueue messages to the table.
- The background case prediction server processes (BGPREDICT) dequeue and process messages from the table.

Structure

The `sw_db_predictqueue_n` table has the following structure:

¹

```
TABLE sw_db_predictqueue_n (
    rowid          NUMERIC(15)          identity(1,1),
    last_failed    NUMERIC(10)          NULL,
    failure_count  INTEGER              NOT NULL,
    msg_id         uniqueidentifier     NOT NULL,
    msg_hdr        VARCHAR(512)         NULL,
    msg_data       VARCHAR(1024)        NOT NULL
    priority       INTEGER              NOT NULL)
```

Column	Description
rowid	Identifier of the row in the table for this message.
last_failed	<p>Number of seconds since January 1st, 1970, when this message last failed to be processed.</p> <p>When this value equals or exceeds the value of the IQL_RETRY_DELAY process attribute, the message is retried.</p>
failure_count	<p>Number of times that this message has failed to be processed.</p> <p>When this value equals or exceeds the value of the IQL_RETRY_COUNT process attribute, the message is moved to the sw_db_deadqueue.</p>
msg_id	Unique identifier of this message.
msg_hdr	Header data associated with this message.
msg_data	Message data.
priority	<p>Message queue priority. The lower this value is, the higher the message queue priority is.</p> <p>The default value is 50.</p>
Note	<p>See "Administering Process Attributes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for more information about the IQL_RETRY_DELAY and IQL_RETRY_COUNT attributes.</p>

Primary Key

No primary key is defined for this table.

Triggers

None.

Indexes

The following clustered index is defined for this table.

Index Name	Indexed Column(s)
idx_sw_db_predictqueue_n ¹	row_id

Table Activity

Each sw_db_predictqueue_n table contains one row for each enqueued message. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	an iProcess process enqueues a message to this table.
updated	a BGPREDICT process dequeues a message from this table but cannot successfully process it (but the message has not exceeded the IQL_RETRY_COUNT value).
deleted	a BGPREDICT process dequeues a message from this table and either: <ul style="list-style-type: none"> successfully processes it. cannot successfully process it, and moves it to the sw_db_deadqueue table because it has exceeded the IQL_RETRY_COUNT value.

¹

sw_db_deadqueue

The `sw_db_deadqueue` table holds failed messages from the `sw_db_bgqueue_n`, `sw_db_wisqueue_n` and `sw_db_predictqueue_n` tables.

Structure

The `sw_db_deadqueue` table has the following structure:

```
TABLE sw_db_deadqueue (
  failed_by      varchar(64)          NOT NULL
  rowid          NUMERIC(15)          identity(1,1),
  last_failed    NUMERIC(10)          NULL,
  failure_count  INTEGER              NOT NULL,
  msg_id         uniqueidentifier     NOT NULL,
  msg_hdr        VARCHAR(512)         NULL,
  msg_data       VARCHAR(1024)        NOT NULL
  priority       INTEGER              NOT NULL)
```

Column	Description
failed_by	Identifies the queue table that this message originates from. One of the following processes: BG (for a message from a <code>sw_db_bgqueue_n</code> table). WIS (for a message from a <code>sw_db_wisqueue_n</code> table). BGPREDICT (for a message from a <code>sw_db_predictqueue_n</code> table).
rowid	Identifier of the row in the table for this message.
last_failed	Number of seconds since January 1st, 1970, when this message last failed to be processed.
failure_count	Number of times that this message has failed to be processed. Note: Messages in this table are not retried.
msg_id	Unique identifier of this message.

Column	Description
msg_hdr	Header data associated with this message.
msg_data	Message data.
priority	Message queue priority. The lower this value is, the higher the message queue priority is. The default value is 50.

Primary Key

No primary key is defined for this table.

Triggers

None.

Indexes

The following clustered index is defined for this table.

Index Name	Indexed Column(s)
idx_sw_db_deadqueue ¹	row_id

Table Activity

The sw_db_deadqueue table contains one row for each message that has exceeded its IQL_RETRY_COUNT threshold value.

Rows are added, updated and deleted in the following situations.

¹

A row is...	When...
added	a message is moved to this table from a sw_db_bgqueue_n , sw_db_wisqueue_n or sw_db_predictqueue_n table, because it has exceeded the IQL_RETRY_COUNT value.
updated	never.
deleted	never.

Creating Additional SQL Database Queue Tables

If you decide to add an additional message queue to your system, you need to:

1. manually create the database queue table needed to hold the new message queue.
2. create the new message queue and map the database queue table to it (using the `SWDIR\bin\swadm` utility).
3. add the message queue to the appropriate Mbox set (using the `SWDIR\bin\swadm` utility).

Each individual message queue must be held in its own database queue table.

Each database queue table *must* have the following characteristics:

- the same column definitions as a [sw_db_bgqueue_n](#), [sw_db_wisqueue_n](#), or [sw_db_predictqueue_n](#) table. (Each of these tables has the same structure.)
- no primary key.
- a clustered index on the `row_id` column.
- the iProcess background user (default `swpro`) must have at least insert, select and delete permissions on the table.
- the iProcess foreground user (default `swuser`) must have at least insert permissions on the table.

**Warning**

If a table that does not conform to these requirements is used as a message queue, messages cannot be enqueued to or dequeued from that queue, and the iProcess Engine may not function correctly.

Example

Suppose that the volume of messages handled by your system has increased significantly, and the default message queues can no longer cope. To deal with the additional load you have decided that you need to add a new BGMBOX3 message queue to the BGMBSET Mboxset. This queue requires a new SQL database queue table `sw_db_bgqueue_3`. You store your database queue tables in the default iProcess database.

To do this:

1. Connect to the SQL Server that holds the iProcess database.
2. Create a new table in the iProcess database called `sw_db_bgqueue_3`.

For example:

go

```
create table swpro.sw_db_bgqueue_3
(
    rowid                NUMERIC(15)                identity(1,1),
    last_failed          NUMERIC(10)                NULL,
    failure_count        INTEGER                    NOT NULL,
    last_failed          NUMERIC(10)                NULL,
    failure_count        INTEGER                    NOT NULL,
    msg_id               uniqueidentifier           NOT NULL,
    msg_hdr              VARCHAR(512)               NULL,
    msg_data             VARCHAR(1024)              NOT NULL
);
```

go

grant references, select, insert, delete, update on

swpro.sw_db_bgqueue_3 to swpro,swuser

go

create clustered index idx_sw_db_bgqueue_3 on swpro.sw_db_bgqueue_3 (rowid);

3. Use the `SWDIR\util\swadm` utility to add a new message queue called BGMBOX3, which uses the `sw_db_bgqueue_3` queue table.

```
cd SWDIR\util
swadm ADD_QUEUE BGMBOX3 Local 0003:swpro.sw_db_bgqueue_3
```

4. Add the BGMBOX3 queue to the BGMBSET Mbox set.

```
swadm ADD_QUEUE_TO_MBOXSET 1 8
```

1 is the number of the BGMBSET Mboxset (from the `swadm SHOW_MBOXSETS` command), and 8 is the number of the message queue (from the `swadm SHOW_QUEUES` command).

Sequence Numbers

This section describes *sequence numbers* - unique numbers that are used by TIBCO iProcess® Engine server processes, and the table that is used to generate them.

About Sequence Numbers

A *sequence number* is simply a unique identifier for an object. TIBCO iProcess® Engine uses six different types of sequence number, as shown in the following table.

Sequence Number	Stored in table...	Unique identifier for a...
o_reqid	staffo	Work item
casenum	case_information	Case
proc_id	proc_index	Procedure
wait_id	wait	Outstanding Wait
def_id	cdqp_def	CDQP definition
cfg_id	cdqp_cfg	CDQP value
monitor_id	iap_monitor	Procedure that IAP is monitoring
provider_id	eaiws_jms_provider ¹	JMS provider
destination_id	eaiws_jms_destination ²	JMS endpoints for JMS provider

¹Only created if TIBCO iProcess Technology Plug-ins are installed.

²

These sequence numbers are generated on an “as required” basis by iProcess Engine, which calls one of the following stored database procedures:

- `sp_cdqp_cfg_sequence`
- `sp_cdqp_def_sequence`
- `sp_cnum_sequence`
- `sp_procid_sequence`
- `sp_reqid_sequence`
- `sp_waitid_sequence`
- `sp_iap_monitor_id_sequence`
- `sp_eaiws_jms_provider_seq`
- `sp_eaiws_jms_destination_seq`

The procedure accesses the `sequences` table, increments the value of the `seq_val` column for the appropriate row, identified by the `seq_id` column, and returns that value. The returned value is then used as the next sequence number in the appropriate table.

**Note**

For more information about these stored procedures please see the database creation script (`init2Ksql.sql`).

However, getting sequence numbers directly from the database in this way can create a performance bottleneck, because while one process is requesting a number it must block any other process from attempting to do so.

To minimize the effect of this bottleneck, you can assign a cache of a block of sequence numbers to a process, by using process attributes. The process gets a sequence number from its cache when it needs one, and only accesses the database to refresh the cache when it has run out of numbers. For more information, see "Sequence Caching" in *TIBCO iProcess Engine Administrator's Guide*.

Table Relationships

The [sequences](#) table has no trigger-enforced relationships with other tables.

sequences

The sequences table is used to generate unique sequence numbers for the use of TIBCO iProcess® Engine server processes.

Structure

The sequences table has the following structure:

```
TABLE sequences (
    seq_id      INTEGER      NOT NULL,
    seq_val     NUMERIC(20)   NOT NULL,
    seq_name    VARCHAR(24)   NOT NULL)
```

Column	Description
seq_id	Sequence ID of the associated seq_val value. One of the following values: 1 (o_reqid) 2 (casenum) 3 (proc_id) 4 (wait_id) 5 (def_id) 6 (cfg_id)
seq_val	Current sequence number value for the sequence defined by seq_id.
seq_name	Name of the associated seq_id column. One of the following values: REQID CNUM PROC WAIT CDQP_DEF CDQP

Column	Description
Note: This value is not currently used by the iProcess Suite.	

Primary Key

None.

Triggers

None.

Indexes

The following clustered index is defined for this table.

Index Name	Column(s) Indexed
idx_sequences	seq_id

Table Activity

This table always contains 6 rows—one row for each type of sequence number used by the iProcess Engine server processes. The table is populated when the iProcess Engine is installed.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	never.
updated	when a new sequence number of that type is requested.
deleted	never.

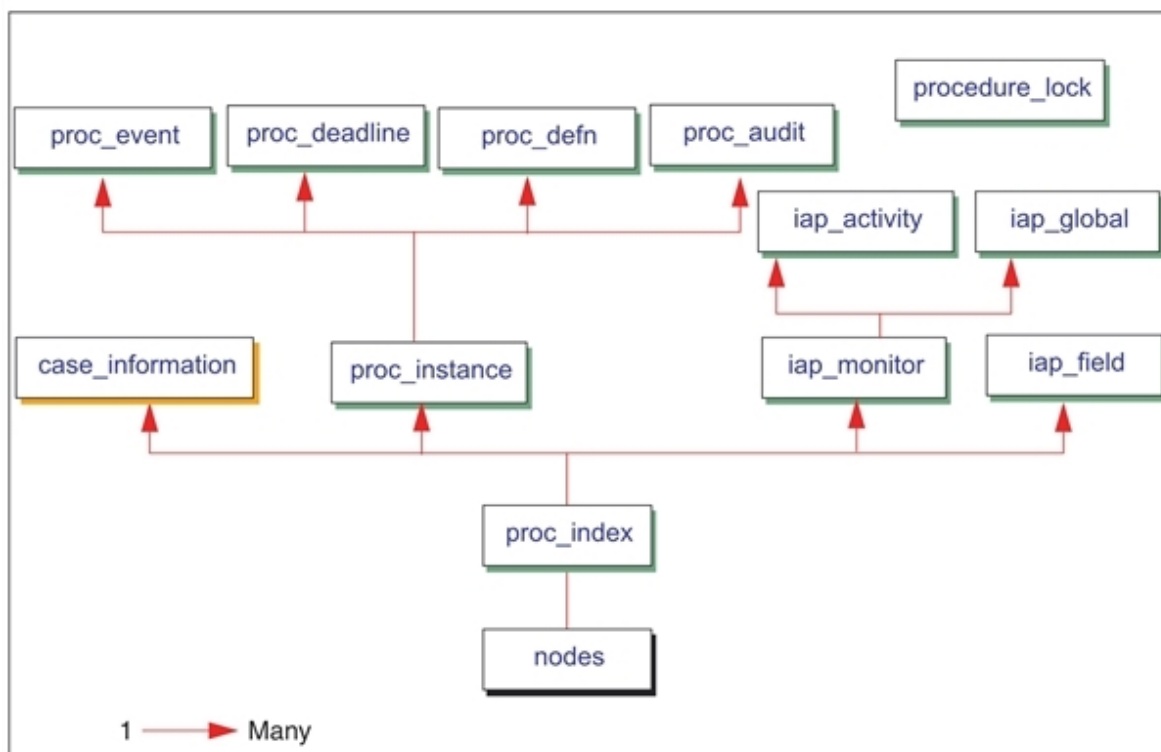
Procedures

This section describes the tables that are used to store information about iProcess procedures, sub-procedures and sub-procedure parameter templates.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



proc_index

The `proc_index` table holds information that is specific to a procedure (or sub-procedure or sub-procedure parameter template).



Note

Data that can change between versions or instances of a procedure is not held in this table. See the [proc_version](#) and [proc_instance](#) tables instead.

Structure

The `proc_index` table has the following structure:

```
TABLE proc_index (
  node_id          INTEGER          NOT NULL,
  proc_id          INTEGER          NOT NULL,
  proc_used_count  SMALLINT         NOT NULL,
  proc_name        VARCHAR(8)       NULL,
  proc_desc        VARCHAR(24)      NULL,
  proc_owner       VARCHAR(49)      NULL,
  dir_name         VARCHAR(12)      NULL,
  proc_used        SMALLINT         NOT NULL,
  work_days        SMALLINT         NOT NULL,
  auto_purge       SMALLINT         NOT NULL,
  networked        SMALLINT         NOT NULL,
  cdesc_type       SMALLINT         NOT NULL,
  ignore_blanks    SMALLINT         NOT NULL,
  is_predict       SMALLINT         NOT NULL,
  normalise_data   SMALLINT         NOT NULL,
  delay_purge      SMALLINT         NOT NULL,
  delay_value      VARCHAR(512)     NULL)
```

Column	Description
<code>node_id</code>	ID of the node that this procedure is defined on, as defined in the nodes table.
<code>proc_id</code>	Unique ID of this procedure, generated from the sequences table.
<code>proc_used_</code>	Not used.

Column	Description
count	
proc_name	Name of this procedure. Note: Internal procedures are prefixed with a dollar sign (\$) character.
proc_desc	Description of this procedure.
proc_owner	Name of the owner of this procedure, as defined in the user_names table.
dir_name	Not used.
proc_used	Flag that defines whether this record is currently free (0) or being used (1).
work_days	Flag that defines whether the procedure uses a 7-day week (0) or a configurable working week (1) in date calculations.
auto_purge	Flag that defines whether (1) or not (0) cases of this procedure are automatically purged when they are closed.
networked	<i>Reserved for possible future use.</i>
cdesc_type	Flag that defines whether a case description is Required (0), Optional (1) or Hidden (2) when a case of this procedure is started.
ignore_blanks	Flag that defines whether or not a blank field is treated as an error when used as an addressee for a step of this procedure: <ul style="list-style-type: none"> 0 means that the field is treated as an error and the step is delivered to the undelivered queue. 1 means that the field is not treated as an error.
is_predict	Flag that defines whether (1) or not (0) case prediction is enabled for this procedure.
normalise_data	Flag that defines whether (1) or not (0) case data normalization is enabled for this procedure.

Column	Description
delay_purge	Flag that defines whether or not to delay the auto-purge operation. <ul style="list-style-type: none"> • 0 means that the auto-purge operation is not delayed. • 1 means that the auto-purge operation is delayed.
delay_value	The value of the delay_value column is: <ul style="list-style-type: none"> • If the value of the delay_purge column is 0, then the value of the delay_value column is NULL. • If the value of the delay_purge column is 1, then the value of the delay_value column is specified in one of the following ways: <ul style="list-style-type: none"> — The period of the delay in days. — Delayed date and time expressions: <i>date expression^time expression</i>

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_proc_index	proc_id node_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_proc_index	DELETE	case_information proc_version proc_instance

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_proc_index_fk	node_id

Table Activity

The `proc_index` table contains one row for each procedure defined on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new procedure is created.
updated	a procedure's details are updated.
deleted	never.

iap_monitor

The `iap_monitor` table holds the monitor ID records for each procedure and node. If a procedure or node has any activity monitoring configured, it is assigned a monitor ID. The monitor ID is then used when correlating between the [iap_activity](#) and [iap_field](#) tables.

Structure

The `iap_monitor` table has the following structure:

```
TABLE iap_monitor(
  node_id      INTEGER      NOT NULL,
  proc_id      INTEGER      NOT NULL,
  monitor_id   numeric(10)   NOT NULL)
```

Column	Description
node_id	ID of the node that this procedure is defined on, as defined in the nodes table.
proc_id	Unique ID of this procedure, generated from the sequences table.
monitor_id	Unique ID of the record for the procedure or node being monitored.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_iap_monitor	monitor_id

Triggers

None.

Indexes

None.

Table Activity

The iap_monitor table contains one row for each procedure or node that has activity monitoring configured for it. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new procedure or node has activity monitoring configured.
updated	a procedure or node's activity monitoring configuration is updated.
deleted	never.

iap_field

The `iap_field` table holds the list of fields that will be published for a given monitor ID.

Structure

The `iap_field` table has the following structure:

```
TABLE iap_field (
    monitor_id      numeric(10)      NOT NULL,
    field_name      VARCHAR(31)     NOT NULL)
```

Column	Description
<code>monitor_id</code>	Unique ID of the record for the procedure or node being monitored, as defined in the iap_monitor table.
<code>field_name</code>	The name of the iProcess Engine field for which data is to be sent out with the activity event.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
<code>pk_iap_field</code>	<code>monitor_id</code> <code>field_name</code>

Triggers

None.

Indexes

None.

.

Table Activity

The `iap_field` table contains one row for each field that will be published for every activity. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new field is created.
updated	a field's details are updated.
deleted	never.

iap_activity

The `iap_activity` table holds the activity and steps which are configured for a given monitor record.

Structure

The `iap_activity` table has the following structure:

```
TABLE iap_activity(
    monitor_id    numeric(10)    NOT NULL,
    activity_id   numeric(3)     NOT NULL,
    step_name     varchar(8)     NOT NULL)
```

Column	Description
<code>monitor_id</code>	Unique ID of the record for the procedure or node being monitored, as defined in the iap_monitor table.
<code>activity_id</code>	Unique ID which represents the activity that is being monitored on the specified iProcess Engine procedure or step.
<code>step_name</code>	The name of the step in the procedure to be monitored. If the step name is \$ALL\$, it means every step in the procedure.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_iap_activity	monitor_id activity_id step_name

Triggers

None.

Indexes

None.

Table Activity

The `iap_activity` table contains one row for each activity that is being monitored on a procedure or node. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new activity to be monitored has been configured for a procedure or node.
updated	an activity's details have been updated for a procedure or node.
deleted	never.

iap_global

The `iap_global` table holds the fields that have been allocated globally to the specified procedure.

Structure

The iap_global table has the following structure:

```
TABLE iap_global(
    node_id          INTEGER          NOT NULL,
    proc_id          INTEGER          NOT NULL,
    field_name       VARCHAR(31)      NOT NULL)
```

Column	Description
node_id	ID of the node that this procedure is stored on, as defined in the nodes table.
proc_id	Unique ID of this procedure, generated from the sequences table.
field_name	The name of the iProcess Engine field for which data is to be sent out with the activity event.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_iap_global	proc_id node_id field_name

Triggers

None.

Indexes

None.

Table Activity

The `iap_global` table contains one row for each field that has been allocated globally to the specified procedure. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new global field has been allocated to the specified procedure.
updated	a global field's details have been updated.
deleted	never.

proc_version

The `proc_version` table holds information that is specific to a version of a procedure (or sub-procedure or sub-procedure parameter template).



Note

Data that is specific to a procedure or to an instance of a procedure is not held in this table. See the [proc_index](#) and [proc_instance](#) tables instead.

Structure

The `proc_version` table has the following structure:

```
TABLE proc_version(
  node_id          INTEGER          NOT NULL,
  proc_id          INTEGER          NOT NULL,
  major_vers       SMALLINT         NOT NULL,
  minor_vers       SMALLINT         NOT NULL,
  pd_version       INTEGER          NULL,
  pv_status        SMALLINT         NULL,
  pv_user          VARCHAR(49)      NULL,
  pv_comment       VARCHAR(128)     NULL,
  pv_created       DATETIME         NULL,
  pv_modified      DATETIME         NULL,
  pv_released      DATETIME         NULL,
  pv_withdrawn     DATETIME         NULL,
  pv_is_subproc    SMALLINT         NULL)
```

Column	Description
node_id	ID of the node that this procedure is stored on, as defined in the nodes table.
proc_id	Procedure number of the procedure associated with this version, as defined in the proc_index table.
major_vers	Major version number of this version.
minor_vers	Minor version number of this version.
pd_version	Instance number of the procedure definition that corresponds to this version, as defined in the proc_instance table.
pv_status	Status of this version. Either: Released (0), Incomplete (1), Unreleased (2), Model (3) or Withdrawn (14).
pv_user	Name of the user who created this version, as defined in the user_names table.
pv_comment	Comment describing this version.
pv_created	Date and time that this version was created.
pv_modified	Date and time that this version was last modified.
pv_released	Date and time that this version was released.
pv_withdrawn	Date and time that this version was withdrawn.
pv_is_subproc	Flag that defines whether (1) or not (0) this version is a sub-procedure.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_proc_version	node_id proc_id major_vers minor_vers

Triggers

None.

Indexes

None.

Table Activity

The `proc_version` table contains one row for every version of every procedure (or sub-procedure or sub-procedure parameter template) defined on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new procedure or version is created.
updated	a version's details are updated.
deleted	a procedure or version is deleted.

procedure_lock

The `procedure_lock` table holds the locks that are used to control access to procedures.

Structure

The `procedure_lock` table has the following structure:

```
TABLE procedure_lock (
    node_id      INTEGER          NOT NULL,
    proc_id      INTEGER          NOT NULL,
    lock_state   SMALLINT         NOT NULL,
    lock_owner   VARCHAR(24) NULL,
    lock_date    DATETIME         NOT NULL,
    lock_reason  SMALLINT         NOT NULL)
```

Column	Description
node_id	ID of the node that this procedure is defined on, as defined in the nodes table.
proc_id	ID of this procedure, as defined in the proc_index table.
lock_state	Flag that defines the procedure state: either unlocked (0) or locked (1).
lock_owner	Name of the user who has the procedure definition locked (if lock_state = 1), as defined in the user_names table.
lock_date	Date and time when the procedure lock was created.
lock_reason	Defines why the procedure is locked: <ul style="list-style-type: none"> • 0 not locked. • 1 locked by the TIBCO iProcess Modeler. • 2 locked by <i>SWDIR\bin\swutil</i> IMPORT.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_procedure_lock	proc_id node_id

Triggers

None.

Indexes

None.

Table Activity

The `procedure_lock` table contains one row for each procedure on the system that is currently being edited.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a procedure is opened (for example, in the iProcess Modeler).
updated	Never.
deleted	a procedure is closed (for example, in the iProcess Modeler).

proc_instance

The `proc_instance` table holds information that is specific to an instance of a version of a procedure (or sub-procedure or sub-procedure parameter template).



Note

Data that is specific to a procedure or to a version of a procedure is not held in this table. See the [proc_index](#) and [proc_version](#) tables instead.

Structure

The `proc_instance` table has the following structure:

```
TABLE proc_instance(
  node_id          INTEGER          NOT NULL,
```

proc_id	INTEGER	NOT NULL,
major_vers	SMALLINT	NOT NULL,
minor_vers	SMALLINT	NOT NULL,
pd_version	INTEGER	NOT NULL,
pi_first_step	VARCHAR(8)	NULL,
pi_rpa_start	SMALLINT	NULL,
pi_rpa_admin	SMALLINT	NULL,
pi_has_eis_objs	SMALLINT	NULL,
pi_has_subprocs	SMALLINT	NULL)

Column	Description
node_id	ID of the node that this instance is stored on, as defined in the nodes table.
proc_id	Procedure number of the procedure associated with this instance, as defined in the proc_index table.
major_vers	Major version number of the version associated with this instance, as defined in the proc_version table.
minor_vers	Minor version number of the version associated with this instance, as defined in the proc_version table.
pd_version	Instance number of this procedure definition.
pi_first_step	Start step for this instance of the procedure.
pi_rpa_start	Flag that defines whether (1) or not (0) Remote Procedure Access (RPA) case start restrictions are set in the procedure definition.
pi_rpa_admin	Flag that defines whether (1) or not (0) Remote Procedure Access (RPA) administration restrictions are set in the procedure definition.
pi_has_eis_objs	Flag that defines whether (1) or not (0) the procedure definition contains EIS objects.
pi_has_subprocs	Flag that defines whether (1) or not (0) the procedure definition contains sub-procedure steps or graft steps.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_proc_instance	node_id proc_id pd_version

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_proc_instance	DELETE	proc_audit proc_defn

Indexes

None.

Table Activity

The `proc_instance` table contains one row for each instance of each procedure (or sub-procedure or sub-procedure parameter template) defined on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new procedure or version is created, or when an existing procedure definition is edited.
updated	never.
deleted	a procedure or version is deleted.

proc_audit

The `proc_audit` table stores audit events for a version of a procedure (or sub-procedure or sub-procedure parameter template). An audit event occurs whenever:

- a version is created, updated, released or withdrawn,
- the procedure definition instance associated with the version is updated. (For example, when a user makes changes to the procedure definition in the iProcess Modeler but does not change the version number).

Structure

The `proc_audit` table has the following structure:

```
TABLE proc_audit(
  node_id          INTEGER          NOT NULL,
  proc_id          INTEGER          NOT NULL,
  major_vers       SMALLINT         NOT NULL,
  minor_vers       SMALLINT         NOT NULL,
  pd_version       SMALLINT         NULL,
  pa_comment       VARCHAR(128)     NULL,
  pa_event         SMALLINT         NULL,
  pa_date          DATETIME         NULL,
  pa_user          VARCHAR(24)      NULL)
```

Column	Description
<code>node_id</code>	ID of the node that this audit event is stored on, as defined in the nodes table.
<code>proc_id</code>	Procedure number of the procedure associated with this audit event, as defined in the proc_index table.
<code>major_vers</code>	Major version number of the version associated with this audit event, as defined in the proc_version table.
<code>minor_vers</code>	Minor version number of the version associated with this audit event, as defined in the proc_version table.
<code>pd_</code>	Instance number of the procedure definition associated with this audit event, as

Column	Description
version	defined in the proc_instance table.
pa_ comment	Comment describing the audit event.
pa_event	The audit event that occurred. Either: Created (0), Updated (1), Released (2) or Withdrawn (3).
pa_date	Date and time that the audit event occurred.
pa_user	Name of the user who performed the audit event, as defined in the user_names table.

Primary Key

None.

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_proc_audit_fk	node_id proc_id

Table Activity

The `proc_audit` table contains one row for every audit event for every version of every procedure (or sub-procedure or sub-procedure parameter template) defined on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a version is created, updated, released or withdrawn.
updated	never.
deleted	a procedure or version is deleted.

proc_defn

The proc_defn table holds procedure definitions.

Structure

The proc_defn table has the following structure:

```
TABLE proc_defn (
    node_id          INTEGER          NOT NULL,
    proc_id          INTEGER          NOT NULL,
    pd_version       INTEGER          NOT NULL,
    pd_type          INTEGER          NOT NULL,
    pd_index         INTEGER          NOT NULL,
    pd_size          INTEGER          NOT NULL,
    pd_directory     VARCHAR(25)     NOT NULL,
    pd_file          VARCHAR(25)     NOT NULL,
    pd_data          VARBINARY(MAX)  NOT NULL)
```

Column	Description
node_id	ID of the node that this procedure definition is defined on, as defined in the nodes table.
proc_id	ID of the procedure that this procedure definition relates to, as defined in the proc_index table.
pd_version	ID of the procedure instance that this row relates to, as defined in the proc_instance table.

Column	Description
pd_type	Type of procedure definition data stored in this row. Either: <ul style="list-style-type: none"> • 0 <i>pro</i> data (textual procedure definition) • 1 <i>lst</i> data (binary procedure definition) • 2 <i>nod</i> data (not used) • 3 <i>gwd</i> data (iProcess Modeler layout information) • 4 <i>nod</i> data (not used) • 5 VBA project data (VBA project files)
pd_index	<p>Index number into the set of rows that make up this procedure definition.</p> <p>If the procedure definition is longer than 30,000 bytes, multiple rows (in 30,000 byte chunks) are used to store the data. Each segment of the procedure definition data is uniquely identified by its <code>pd_index</code> value.</p> <p>Note:</p> <ul style="list-style-type: none"> • If you have upgraded your system from a pre-Version i10.0-x(3.0) iProcess Engine, existing procedure definitions are divided into 2000 byte chunks. • However, if a procedure definition is modified, the new instance is added using 30,000 byte chunks.
pd_size	Size (in bytes) of the procedure definition data for the current row. This is 30,000 bytes (or 2000—see above) for all but the last row of the procedure definition.
pd_directory	<p>If <code>pd_type</code> is 5, contains the sub-directory path (relative to <i>SWDIR\projects</i>) where any VBA project files related to this procedure definition are stored. Filenames are stored in <code>pd_file</code>.</p> <p>If <code>pd_type</code> is 0 to 4, this field contains a hyphen.</p>
pd_file	If <code>pd_type</code> is 5, contains the filename of a VBA project file related to this procedure definition (if there is one). The file is physically stored in the location defined by <code>pd_directory</code> .

Column	Description
	If pd_type is 0 to 4, this field contains a hyphen.
pd_data	Raw data for this (portion of the) procedure definition.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_proc_defn	proc_id pd_index pd_type node_id pd_version pd_directory pd_file

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_proc_defn	proc_id pd_version pd_type node_id

Table Activity

The proc_defn table contains one or more rows for each instance of each procedure definition on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a procedure is saved in the iProcess Modeler (thus creating a new instance), or imported.
updated	never.
deleted	a procedure is deleted.

proc_deadline

The `proc_deadline` table stores definitions of procedure deadlines.

Structure

The `proc_deadline` table has the following structure:

```
TABLE proc_deadline(
  node_id      INTEGER      NOT NULL,
  proc_id      INTEGER      NOT NULL,
  major_vers   SMALLINT     NOT NULL,
  minor_vers   SMALLINT     NOT NULL,
  pd_version   INTEGER      NOT NULL,
  dead_name    VARCHAR(32)   NOT NULL,
  event_name   VARCHAR(32)   NOT NULL,
  dead_value   VARCHAR(512)  NOT NULL)
```

Column	Description
<code>node_id</code>	ID of the node that this procedure is defined on, as defined in the nodes table.
<code>proc_id</code>	Unique ID of this procedure, generated from the sequences table.
<code>major_vers</code>	The major version number of the procedure that this case belongs to, as defined in the proc_version table.

Column	Description
minor_ vers	The minor version number of the procedure that this case belongs to, as defined in the proc_version table.
pd_ version	Instance number of the procedure definition, as defined in the proc_instance table.
dead_name	The name of the case deadline.
event_ name	The name of the event step that is triggered when the case deadline expires.
dead_ value	<p>The value of the case deadline. The value is specified in one of the following formats:</p> <ul style="list-style-type: none"> If the case deadline is specified as a period, then the value is in the format: <i>minutes^hours^days^weeks^months^years</i> If the case deadline is specified as an expression, then the value is in the format: <i>date expression^time expression</i>

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_proc_deadline	node_id proc_id pd_version dead_name

Triggers

None.

Indexes

The following index is defined for this table.

Key Name	Column(s) Indexed
idx_proc_dl_fk	node_id pro_id pd_version

Table Activity

The proc_deadline table contains one or more rows for each instance of each procedure definition on the system. Rows are added, updated, and deleted in the following situations.

A row is...	When...
added	The deadlines are created.
updated	The deadlines are updated.
deleted	The deadlines are deleted.

proc_event

The proc_event table stores definitions of procedure events.

Structure

The proc_event table has the following structure:

```
TABLE proc_event(
    node_id          INTEGER          NOT NULL,
    proc_id          INTEGER          NOT NULL,
    major_vers       INTEGER          NOT NULL,
    minor_vers       INTEGER          NOT NULL,
    pd_version       INTEGER          NOT NULL,
```

eventname	VARCHAR_TYPE(32)	NOT NULL,
user_event_name	VARCHAR_TYPE(32)	NOT NULL)

Column	Description
node_id	ID of the node that this case is hosted on, as defined in the nodes table.
proc_id	ID of the procedure that this event belongs to, as defined in the proc_index table.
major_vers	Major version number of the procedure version that this case belongs to, as defined in the proc_version table.
minor_vers	Minor version number of the procedure version that this case belongs to, as defined in the proc_version table.
pd_version	Instance number of the procedure definition, as defined in the proc_instance table.
eventname	<p>The name of the procedure event. The value of this column is one of the following:</p> <ul style="list-style-type: none"> • BeforePurge • BeforeClose • AfterClose • BeforeResurrect • AfterResurrect • BeforeSuspend • AfterSuspend • BeforeResume • AfterResume
user_event_name	The name of the event step which you set for the procedure event.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_proc_event	node_id proc_id pd_version eventname

Triggers

None.

Index

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_proc_event_fk	node_id proc_id pd_version

Table Activity

The proc_event table contains one or more rows for each instance of each procedure definition on the system. Rows are added, updated, and deleted in the following situations.

A row is...	When...
added	<p>If one of the following conditions occurred:</p> <ul style="list-style-type: none"> • A new procedure event is added. • A procedure event is modified.

A row is...	When...
	<p>Note: When updating a procedure event, the record related to this event is deleted and then a new record with the event changes is added in the table.</p> <ul style="list-style-type: none"> A new version of a procedure is released.
updated	Never.
deleted	<p>If one of the following conditions occurred:</p> <ul style="list-style-type: none"> A procedure event is deleted. A procedure event is modified. <p>Note: When updating a procedure event, the record related to this event is deleted and then a new record with the event changes is added in the table.</p> <ul style="list-style-type: none"> The version of this procedure is deleted.

wqd_delta_subscriptions

The wqd_delta_subscriptions table holds a list of the work queues, JMS topics and WQDIDs that are currently in use for Work Queue Delta subscriptions published via JMS. It provides a permanent store of subscription details if a WIS process is restarted. See *TIBCO iProcess Engine Administrator's Guide* for details of Work Queue Delta publication via JMS.

Structure

The wqd_delta_subscriptions table has the following structure:

```
TABLE wqd_delta_subscriptions(
    wis_process_instance    numeric(5)           NOT NULL,
    queue_name              varchar(51)         NOT NULL,
    wqid                    varchar(36)          NOT NULL,
    jms_topic_name          varchar(1024)
```

Column	Description
wis_process_instance	The instance of the WIS process that is responding to Work Queue Delta publication requests.
queue_name	The name of the work queue being monitored.
wqdid	The unique ID of the subscription.
jms_topic_name	The name of the JMS topic being used for publication.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_wqd_delta_sub	wqdid

Triggers

None.

Indexes

None.

Procedure Management

This section describes the tables that are used to store information about the iProcess procedure objects that are stored in the Procedure Management library.

About Procedure Objects

Information is stored in these tables about the following types of procedure object:

- libraries
- procedures
- sub-procedures
- sub-procedure parameter templates
- shortcuts.

**Note**

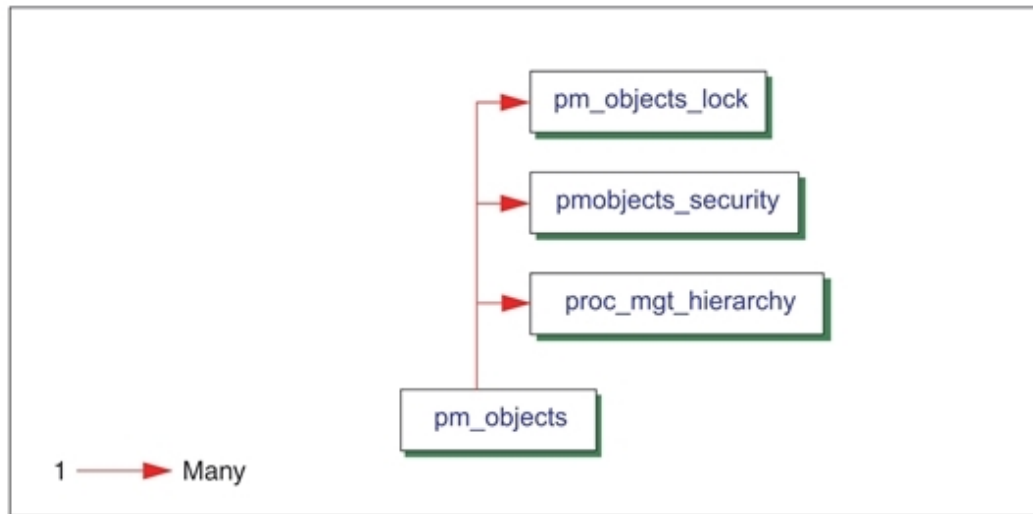
Shortcuts are *not* real procedure objects. They are simply placeholders that allow you to access a procedure object from different locations in the Procedure Management library. Data on shortcuts is only stored in the [proc_mgt_hierarchy](#) table.

Information about procedure versions is stored in other tables - see [Procedures](#) for more information.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



pm_objects

The `pm_objects` stores information about each procedure object (except shortcuts) in the Procedure Management library.

Structure

The `pm_objects` table has the following structure:

```

TABLE pm_objects (
  object_guid          VARCHAR(36)          NOT NULL,
  object_type          SMALLINT              NOT NULL,
  object_name          VARCHAR(64)          NOT NULL,
  version_major        INTEGER NULL,
  version_minor        INTEGER NULL,
  icon_mod_time        DATETIME             NOT NULL,
  icon_binary          VARBINARY(MAX) NULL,
  icon_size            INTEGER              NOT NULL,
  object_url           VARCHAR(1000) NULL,
  author              VARCHAR(64) NULL,
  object_create_time   DATETIME             NOT NULL,
  object_mod_time      DATETIME             NOT NULL,
  release_id          VARCHAR(64) NULL,
  security_all         SMALLINT              NOT NULL,
  proc_id             INTEGER              NOT NULL,
  proc_status         SMALLINT NULL)
  
```

Column	Description
object_guid	Globally unique, system-generated identifier for this procedure object. The row defining the Procedure Management library root has the value ROOT_LIBRARY_GUID.
object_type	Procedure object type. Either: library (0), procedure (1), sub-procedure (2) or sub-procedure parameter template (3).
object_name	Name of this procedure object. Note: The object's description (if defined) is also included as part of this field, in brackets following the name.
version_major	Major version number of the version associated with this procedure object, as defined in the proc_version table. This value is always 0 if the object is a library.
version_minor	Minor version number of the version associated with this procedure object, as defined in the proc_version table. This value is always 0 if the object is a library.
icon_mod_time	Time that the icon associated with this procedure object was last modified.
icon_binary	Binary form of the icon associated with this procedure object.
icon_size	Size (in bytes) of the icon associated with this procedure object.
object_url	Usage URL associated with this procedure object.
author	Value of the Author extended property for this procedure object.
object_create_time	Value of the Date Created extended property for this procedure object, showing the time that this object was created.
object_mod_time	Value of the Date Modified extended property for this procedure object, showing the time that this procedure object was last modified.

Column	Description
release_id	Value of the Release Identification extended property for this procedure object.
security_all	Flag that defines whether (1) or not (0) the OEM lock is set for this procedure object.
proc_id	Procedure number of the procedure associated with this procedure object, as defined in the proc_index table. This value is always -1 if the object is a library.
proc_status	For internal use only.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_pm_objects	object_guid

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_pm_objects	DELETE	pmobjects_security

Indexes

None.

Table Activity

The pm_objects table contains one row for each procedure object (except shortcuts) in the Procedure Management library.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a procedure object is created.
updated	a procedure object is modified.
deleted	a procedure object is deleted.

pm_objects_lock

The pm_objects_lock table stores information about every procedure object that is currently locked.

Structure

The pm_objects_lock table has the following structure:

```
TABLE pm_objects_lock (
  object_guid    VARCHAR(36)      NOT NULL,
  lck_state      INTEGERNULL,
  lck_owner      VARCHAR(24) NULL,
  lck_time       DATETIME NULL)
```

Column	Description
object_guid	ID for this procedure object, as defined in the pm_objects table.
lck_state	Flag that defines (1) that this procedure object is locked.
lck_owner	Name of the user who has this procedure object locked, as defined in the user_names table.
lck_time	Time that the lock was set on this procedure object.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_pm_objects_lock	object_guid

Triggers

None.

Indexes

None.

Table Activity

The pm_objects_lock table contains one row for each procedure object that is currently locked.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a procedure object is locked.
updated	never.
deleted	a procedure object is unlocked.

Unlocking Incorrectly Locked Procedure Objects

A procedure object is normally shown as locked in the Procedure Manager when it is open in the TIBCO iProcess Modeler. However, an object may also be locked if it was not closed properly from a previous TIBCO iProcess Modeler session - for example, if the system failed while the procedure was open.

If this happens the object cannot be accessed again until the locks in the [proc_index](#) and [pm_objects](#) tables are released. To do this:

1. Log in to SQL Server as the background user.
2. In SQL Query Analyzer, use the following query to delete all locks associated with the locked procedure (where *procedure_name* is the name of the locked procedure):

```
delete procedure_lock where proc_id = (select proc_id from proc_index
where proc_name = 'procedure_name')
delete pm_objects_lock where object_guid = (select object_guid from pm_
objects where object_name like 'proc_name%')
```

3. Commit the transaction.

The object should now appear unlocked in Procedure Manager. (You can refresh the display first.)



Note

The like statement requires a % sign prefixed or suffixed to the *procedure_name*. However, this will select similarly-named procedures - for example, 'TEST%' would select procedures named TEST1, TEST2, TEST3, even if only TEST4 needed to be cleared.

You are recommended to ensure that the procedure name is complete - in the case in the previous paragraph, specify 'TEST4%' - so that the % character only covers the option procedure description.

TIBCO also recommend that you ensure you are connected to the correct database and table.

pmobjects_security

The pmobjects_security table stores the encrypted security settings for every procedure object (except shortcuts) in the Procedure Management library.

Structure

The pmobjects_security table has the following structure:

```
TABLE pobjects_security (
    object_guid    VARCHAR(36)          NOT NULL,
    security_id    INTEGER              NOT NULL,
    attrib_expr    VARCHAR(260)         NOT NULL,
    security_level VARCHAR(8)           NOT NULL)
```

Column	Description
object_guid	ID for this procedure object, as defined in the pm_objects table.
security_id	Internal identifier for this procedure object.
attrib_expr	Encrypted security attribute expression for this procedure object.
security_level	Encrypted security level for this procedure object.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_pobjects_security	object_guid security_id

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_pobjects_security_fk	object_guid

Table Activity

The `pmobjects_security` table contains one row for each procedure object (except shortcuts) in the Procedure Management library.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a procedure object is created.
updated	a procedure object's security settings are modified.
deleted	a procedure object is deleted.

proc_mgt_hierarchy

The `proc_mgt_hierarchy` table stores a set of hierarchy records, which define the hierarchical structure of the Procedure Management library. Each record defines the location of a procedure object in the library.

Structure

The `proc_mgt_hierarchy` table has the following structure:

```
TABLE proc_mgt_hierarchy(
  parent_guid      VARCHAR(36)          NOT NULL,
  object_guid      VARCHAR(36)          NOT NULL,
  is_shortcut      SMALLINT              NOT NULL)
```

Column	Description
<code>parent_guid</code>	ID for the parent library, as defined in the pm_objects table.
<code>object_guid</code>	ID for this procedure object, as defined in the pm_objects table.

Column	Description
	Note: If <code>is_shortcut</code> is 1, this value is the identifier of the procedure object pointed to by the shortcut.
<code>is_shortcut</code>	Flag that defines whether this hierarchy record is for a real procedure object (0) or for a shortcut (1).

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
<code>pk_proc_mgt_hier</code>	<code>parent_guid</code> <code>object_guid</code>

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
<code>idx_proc_mgt_hierarchy_fk</code>	<code>object_guid</code>

Table Activity

The `proc_mgt_hierarchy` table contains one row for every procedure object in the Procedure Management library (except for the root Procedure Management library).

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a procedure object is created, copied or moved.
updated	never.
deleted	a procedure object is deleted or moved.

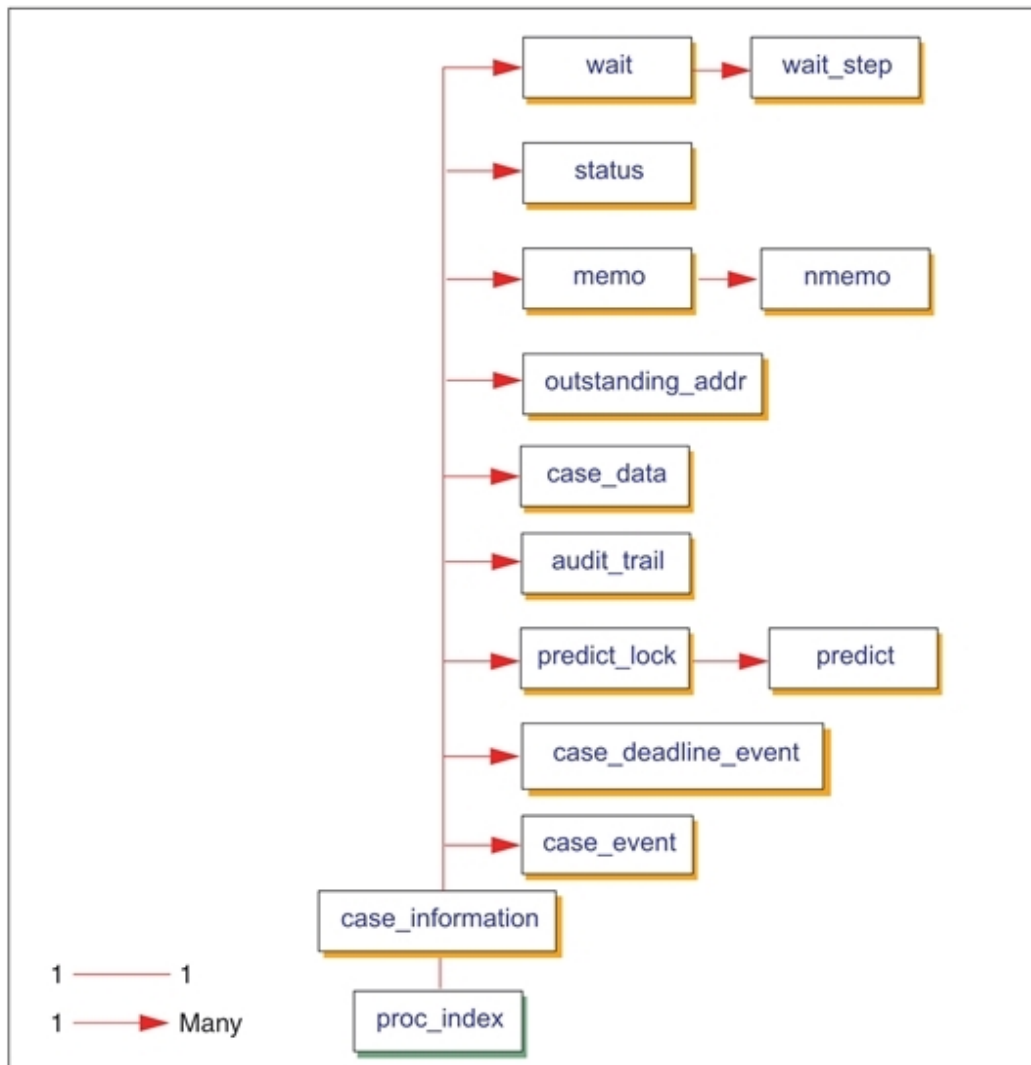
Cases

This section describes the tables that are used to store information about iProcess cases.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



case_information

The `case_information` table holds information about every case and sub-case that has been started and not yet purged on the system.

Structure

The `case_information` table has the following structure:

```
TABLE case_information (
  node_id          INTEGER          NOT NULL,
```

```

proc_id      INTEGER      NOT NULL,
casenum      NUMERIC(20)   NOT NULL,
starter      VARCHAR(49)   NOT NULL,
casedesc     VARCHAR(24)   NULL,
procflags    SMALLINT      NOT NULL,
next_deadline DATETIME NULL,
is_subcase   SMALLINT      NOT NULL,
is_dead      SMALLINT      NOT NULL,
is_suspended SMALLINT      NOT NULL,
major_vers   INTEGER       NOT NULL,
minor_vers   INTEGER       NOT NULL,
proc_precedence INTEGER     NOT NULL,
started      DATETIME      NOT NULL,
started_usecs NUMERIC(10)   NOT NULL,
using_blob   NUMERIC(10)    )

```

Column	Description
node_id	ID of the node that this case is hosted on, as defined in the nodes table.
proc_id	ID of the procedure that this case belongs to, as defined in the proc_index table.
casenum	<p>Unique case number for this case, generated from the sequences table.</p> <p>Note: If the system has been upgraded from a Version 9 Process Engine, any cases that were started before the upgrade do not have a unique case number. (They have a number that is unique to that procedure.)</p>
starter	Name of the user who started this case, as defined in the user_names table.
casedesc	Case description supplied when the case was started.
procflags	<p>The procedure flags that were set at the time the case was started. For internal use only.</p> <p>Note: These flags are stored to allow consistent operation of the case if the procedure changes status during the lifetime of the case. For example, if the procedure is unreleased when the case is started, but changes to released before the case completes, the case can continue using the original procedure flags.</p>

Column	Description
next_deadline	Date and time that the next deadline expires on this case. If no deadline is set this value appears as 12/31/3000 11:15:00 PM.
is_subcase	Flag that defines whether this case is a main case (0) or a sub-case (1).
is_dead	Flag that defines whether (1) or not (0) this case has completed.
is_suspended	Flag that defines whether (1) or not (0) the case is currently suspended (from a TIBCO iProcess Objects or SAL application).
major_vers	Major version number of the version of the procedure that this case belongs to, as defined in the proc_version table.
minor_vers	Minor version number of the version of the procedure that this case belongs to, as defined in the proc_version table.
proc_precedence	Stores the procedure precedence settings for opening sub-cases. One of: <ul style="list-style-type: none"> • 32 - Released only. • 64 - Unreleased > Released. • 96 - Model > Released. • 128 - Unreleased > Model > Released. • 160 - Model > Unreleased > Released.
started	Date and time that the case was started, to the resolution of a second. Note: The started_usecs column can be combined with this column to provide resolution to a microsecond.
started_usecs	Number of microseconds since the start of the <i>seconds</i> value specified in the started column.
using_blob	Decides to use the old memo data table or the new one.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_case_information	casenum proc_id node_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_case_information	DELETE	audit_trail outstanding_addr wait status case_data memo predict_lock

Indexes

The following indexes are defined for this table.

Index Name	Column(s) Indexed
idx_case_information_fk	proc_id node_id

Table Activity

The case_information table contains one row for every open and closed case and sub-case on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new case or sub-case is started.
updated	any of the following occur: <ul style="list-style-type: none"> a case or sub-case is closed. a deadline on a case or sub-case is set or expires. a case or sub-case is suspended or re-opened. a new version of a procedure is released and the option is chosen to migrate cases (and sub-cases) that use the previously released version to the new version.
deleted	a case is purged.

outstanding_addr

The outstanding_addr table holds information about each outstanding step on the system.

Structure

The outstanding_addr table has the following structure:

```
TABLE outstanding_addr (
  rowid UNIQUEIDENTIFIER ROWGUIDCOL NOT NULL,
  node_id INTEGER NOT NULL,
  proc_id INTEGER NOT NULL,
  casenum NUMERIC(20) NOT NULL,
  sentdate DATETIME NOT NULL,
  deadline SMALLINT NOT NULL,
  deadline_expired SMALLINT NOT NULL,
  sub_procedure SMALLINT NOT NULL,
  deaddate DATETIME NOT NULL,
  stepname VARCHAR(8) NOT NULL,
  user_name VARCHAR(64) NOT NULL,
  reqid NUMERIC(20) NOT NULL,
  item_suspended SMALLINT NOT NULL,
```

item_withdrawn array_idx	SMALLINT INTEGER	NOT NULL, NOT NULL)
-----------------------------	---------------------	------------------------

Column	Description
rowid	Unique identifier for this row
node_id	ID of the node that this outstanding step is hosted on, as defined in the nodes table.
proc_id	ID of the procedure that this outstanding step belongs to, as defined in the proc_index table.
casenum	Number of the case that this outstanding step belongs to, as defined in the case_information table.
sentdate	Date and time that this outstanding step was sent to the user_name queue.
deadline	Flag that defines whether (1) or not (0) this outstanding step has a deadline defined.
deadline_expired	Flag that defines whether (1) or not (0) the deadline (if defined) has expired and been processed.
sub_procedure	Flag that defines whether (1) or not (0) this outstanding step is a sub-procedure call.
deaddate	Date and time that the deadline (if defined) expires on this outstanding step. If no deadline is set this value appears as 12/31/3000 11:15:00 PM.
stepname	Stepname of this outstanding step.
user_name	Name of the queue that this outstanding step has been sent to, as defined in the user_names table.
reqid	Unique ID for this work item, generated from the sequences table.
item_	Flag that defines whether (1) or not (0) this outstanding step is currently

Column	Description
suspended	suspended. Note: item_suspended is only set if the case is suspended <i>and</i> the ignore suspend attribute is <i>not</i> set on the step.
item_withdrawn	Flag that defines whether (1) or not (0) this outstanding step is withdrawn.
array_idx	Either: <ul style="list-style-type: none"> The array element index number of the sub-procedure that generated this outstanding step, if the sub-procedure was called from either a graft step or a dynamic sub-procedure call step. -1, otherwise.

Primary Key

None.

Triggers

None.

Indexes

The following index is for this table.

Index Name	Column(s) Indexed
idx_deadline_date	deaddate

The following clustered index is defined for this table.

Index Name	Indexed Column(s)
idx_outstanding_addr	rowid

Table Activity

The `outstanding_addr` table contains one row for each outstanding step on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	<p>a new step is sent out.</p> <p>Note:</p> <ul style="list-style-type: none"> • if a step has multiple addressees one row is added per addressee. • for a dynamic sub-procedure, one row is added per called sub-procedure. • for a graft step, one row is added per grafted sub-procedure or external step.
updated	<p>any of the following occur:</p> <ul style="list-style-type: none"> • a deadline on an outstanding step expires. • a case is suspended or re-opened. • an outstanding step is withdrawn.
deleted	<p>the background processes a release, withdraw, close or purge operation that affects an outstanding step.</p>

wait

The `wait` table holds information about each outstanding wait on the system.

Structure

The `wait` table has the following structure:

```
TABLE wait (
    wait_id      NUMERIC(10)      NOT NULL,
    node_id     INTEGER          NOT NULL,
    proc_id      INTEGER          NOT NULL,
    casenum     NUMERIC(20)      NOT NULL,
```

parentstep	VARCHAR(8)	NULL,
expression	VARCHAR(200)	NULL,
type	SMALLINT	NOT NULL)

Column	Description
wait_id	Unique ID for this wait, generated from the sequences table.
node_id	ID of the node that this wait is hosted on, as defined in the nodes table.
proc_id	ID of the procedure that this wait belongs to, as defined in the proc_index table.
casenum	Case number that this wait belongs to, as defined in the case_information table.
parentstep	Step name of the parent step for this wait.
expression	<i>Not used. Reserved for possible future use.</i>
type	Wait type. Currently the only supported type is a Step wait (1), that is, the step is waiting for one or more other steps to be released.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_wait	wait_id node_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_wait	DELETE	wait_step

Indexes

The following indexes are defined for this table.

Index Name	Column(s) Indexed
idx_wait_fk	casenum proc_id node_id
idx_wait	casenum proc_id

Table Activity

The `wait` table contains one row for each outstanding wait on the system. An associated record exists in the [wait_step](#) table for each step being waited for.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new wait is triggered.
updated	never.
deleted	a wait is processed.

wait_step

The `wait_step` table holds information about each step that is currently being waited for by a wait defined in the [wait](#) table.

Structure

The wait_step table has the following structure:

```
TABLE wait_step (
    node_id      INTEGER      NOT NULL,
    proc_id      INTEGER      NOT NULL,
    wait_id      NUMERIC(10)   NOT NULL,
    step_id      NUMERIC(10)   NOT NULL)
```

Column	Description
node_id	ID of the node that the wait is hosted on, as defined in the nodes table.
proc_id	ID of the procedure that the wait belongs to, as defined in the proc_index table.
wait_id	ID of the wait, as defined in the wait_step table.
step_id	Number of the step that is being waited for, as defined (by the step_num column) in the status table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_wait_step	wait_id step_id

Triggers

None.

Indexes

The following indexes are defined for this table.

Index Name	Column(s) Indexed
idx_wait_step_fk	wait_id node_id
idx_wait_step	wait_id proc_id

Table Activity

The wait_step table contains one row for each for each step currently being waited for. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new wait is triggered.
updated	never.
deleted	a step that is being waited for is released or withdrawn.

status

The status table holds the current status of each step of each case on the system.

Structure

The status table has the following structure:

```
TABLE status (
  node_id      INTEGER      NOT NULL,
  proc_id      INTEGER      NOT NULL,
  casenum      NUMERIC(20)   NOT NULL,
  step_num     INTEGER      NOT NULL,
  step_status  INTEGER      NOT NULL)
```

Column	Description
node_id	ID of the node that this step is hosted on, as defined in the nodes table.
proc_id	ID of the procedure that this step belongs to, as defined in the proc_index table.
casenum	Case number that this step belongs to, as defined in the case_information table.
step_num	Place number for this step (a unique ID that does not change between edits of a procedure). For internal use only.
step_status	Step status. Either: Not processed (0), Released (1), Outstanding (2) or Withdrawn (3).

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_status	casenum proc_id step_num node_id

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_status_fk	casenum proc_id node_id

Table Activity

The status table contains one row for each step of each case (open or closed) on the system. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a step is sent out, or a case is started.
updated	a step's status changes.
deleted	a case is purged.

case_data

The case_data table holds the central copy of the field name and value of each assigned field in each case on the system.

When a work item is sent out to a queue, field data is copied from the [case_data](#) table to the [pack_data](#) table. The client uses the field values in the [pack_data](#) table to fill out the form correctly. When the form is kept any changed fields are updated in the [pack_data](#) table. When a work item is released field data is moved from the [pack_data](#) table to the [case_data](#) table.

Structure

The case_data table has the following structure:

```
TABLE case_data (
    node_id          INTEGER          NOT NULL,
    proc_id          INTEGER          NOT NULL,
    casenum          NUMERIC(20)      NOT NULL,
    field_name       VARCHAR(31)      NOT NULL,
    field_value      VARCHAR(255)     NULL,
    field_value_N    VARCHAR(255)     NULL)
```

Column	Description
node_id	ID of the node that this field is hosted on, as defined in the nodes table.
proc_id	ID of the procedure that this field belongs to, as defined in the proc_index table.
casenum	Case number that this field belongs to, as defined in the case_information table.
field_name	Name of this field.
field_value	Value of this field.
field_value_N	<p>“Normalized” value of the field_value value. That is:</p> <ul style="list-style-type: none"> • Date values are stored as YYYY-MM-DD. • Numeric values are stored as padded strings. • Time and String values are not changed. <p>Note: This value is stored to make case data searching easier, so that the database can do simple string comparisons, instead of having to do type conversions.</p> <p>Case data can be normalized either when installing/upgrading the iProcess Engine, or by using the Case Data Normalization Utility - see <i>TIBCO iProcess Engine Administrator's Guide</i>.</p>

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_case_data	casenum proc_id node_id field_name

Triggers

None.

Indexes

The following indexes are defined for this table.

Index Name	Column(s) Indexed
idx_case_data_fk	casenum proc_id node_id
idx_case_data_cnum_procid_fname_fvalue ¹	field_name field_value_N casenum proc_id

Table Activity

The case_data table contains n rows for each open case on the system, where n is the number of fields in the case that have assigned data values. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a field has a value assigned to it.
updated	a field's value is changed.
deleted	a field becomes unassigned (blank) or when the parent case is purged.

¹This index can impact purge performance. If a large number of purges are being made at the same time TIBCO recommends that you delete this index before performing the purge, then recreate it when the purge has completed.

audit_trail

The audit_trail table holds information about each event that has happened to each case on the system.

Structure

The audit_trail table has the following structure:

```
TABLE audit_trail (
    node_id          INTEGER          NOT NULL,
    proc_id          INTEGER          NOT NULL,
    casenum           NUMERIC(20)      NOT NULL,
    type_id           INTEGER          NOT NULL,
    audit_date        DATETIME         NOT NULL,
    stepdesc          VARCHAR(24)      NULL,
    user_name         VARCHAR(64)      NULL,
    stepname          VARCHAR(8)       NULL,
    audit_usecs       NUMERIC(10)      NOT NULL,
    major_vers        INTEGER          NOT NULL,
    minor_vers        INTEGER          NOT NULL)
```

Column	Description
node_id	ID of the node that this audit event is hosted on, as defined in the nodes table.
proc_id	ID of the procedure that this audit event belongs to, as defined in the proc_index table.
casenum	Case number that this audit event belongs to, as defined in the case_information table.
type_id	ID of the audit event that occurred. Either: <ul style="list-style-type: none"> a system-defined audit event (<=255), as defined in the <code>SWDIR\etc\language.lng\audit.mes</code> file. a custom, application-defined event (256-999), as defined in the <code>SWDIR\etc\language.lng\auditusr.mes</code> file. <p>Note: See "Defining Audit Trail Entries" in <i>TIBCO iProcess swutil and swbatch</i></p>

Column	Description
	<i>Reference Guide</i> for more information about system-defined and application-defined audit trail entries.
audit_date	<p>Date and time that this audit event occurred.</p> <p>Note: The audit_usecs column can be combined with this column to provide resolution to a microsecond.</p>
stepdesc	<p>If type_id is:</p> <ul style="list-style-type: none"> • ≤ 255, the step description of the step that this audit event occurred to. • ≥ 256, a user-defined string, containing for example the description of this audit event.
user_name	<p>If type_id is:</p> <ul style="list-style-type: none"> • ≤ 255, the name of the user who performed this audit event, as defined in the user_names table. • ≥ 256, a user-defined string, containing for example the name of the user who performed this audit event.
stepname	<p>Name of the step that this audit event occurred for.</p> <p>For internal use only.</p>
audit_usecs	<p>Number of microseconds since the start of the <i>seconds</i> value specified in the audit_date column.</p> <p>Note: On systems that have been upgraded from Version 9, for any existing cases the 6 least significant digits of the at_id column are copied into this column to ensure that audit trail entries remain in the correct order.</p>
major_vers	Major version number of the version of the procedure that this audit event belongs to, as defined in the proc_version table.
minor_vers	Minor version number of the version of the procedure that this audit event belongs to, as defined in the proc_version table.

Primary Key

None.

Triggers

None.

Indexes

The following clustered index is defined for this table.

Index Name	Column(s) Indexed
idx_audit_trail_fk	casenum proc_id node_id

Table Activity

The `audit_trail` table contains one or more rows for each step of each case on the system. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	an audit event occurs.
updated	never.
deleted	a case is purged.

memo

The `memo` table stores the case memo data before upgrading to iProcess Engine 11.6. Since iProcess Engine 11.6, all the new case memo data is stored in the `VARBINARY(MAX)` data type in the `nmemo` table.

You can migrate case memo data in the `memo` table to the `nmemo` table when upgrading to iProcess Engine 11.6 or migrate them by using the `swutil MIGRATEMEMOS` command after the upgrades.



Note

A copy of a memo is kept in the `pack_memo` table if the memo is marked on an outstanding form.

Structure

The `memo` table has the following structure:

```
TABLE memo (
  node_id      INTEGER          NOT NULL,
  proc_id      INTEGER          NOT NULL,
  casenum      NUMERIC(20)      NOT NULL,
  memo_id      INTEGER          NOT NULL,
  memo_index   INTEGER          NOT NULL,
  memo_size    INTEGER          NOT NULL,
  memo_data    VARBINARY(MAX)   NOT NULL,
  array_idx    INTEGER          NOT NULL)
```

Column	Description
<code>node_id</code>	ID of the node that this memo is hosted on, as defined in the nodes table.
<code>proc_id</code>	ID of the procedure that this memo belongs to, as defined in the proc_index table.
<code>casenum</code>	Case number that this memo belongs to, as defined in the case_information table.
<code>memo_id</code>	Unique (for this case) ID of this memo.
<code>memo_index</code>	Index number into the set of rows that make up this memo. If a memo is longer than 30,000 bytes multiple rows (in 30,000 byte chunks) are used to store the memo data. Each segment of the memo data is uniquely identified by its <code>memo_index</code> value.

Column	Description
	<p>Note:</p> <ul style="list-style-type: none"> If you have upgraded your system from a pre-Version i10.0-x(3.0) , existing memos are divided into 2,000 byte chunks. <p>However, if a memo is modified, the existing rows are deleted and then re-added using 30,000 byte chunks.</p>
memo_size	Size (in bytes) of the memo data for this row.
memo_data	Memo data.
array_idx	<p>Either:</p> <ul style="list-style-type: none"> The array element index number of the memo. -1, if the memo is not an array memo field.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_memo	casenum memo_id memo_index proc_id node_id array_idx

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_memo_fk	casenum proc_id node_id

Table Activity

The `memo` table contains one or more rows for each memo on the system. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	either: <ul style="list-style-type: none"> a memo field is first assigned. a memo field is modified. (All rows for the memo are deleted and then re-added.)
updated	never.
deleted	either: <ul style="list-style-type: none"> a memo field is modified. (All rows for the memo are deleted and then re-added.) a case is purged.

nmemo

The `nmemo` table stores the case memo data after upgrading to iProcess Engine 11.6. Since iProcess Engine 11.6, all the new case memo data is stored in the data type in the `nmemo` table.

You can migrate case memo data in the `memo` table to the `nmemo` table when upgrading to iProcess Engine 11.6 or migrate them by using the `swutil MIGRATEMEMOS` command after the upgrades.

**Note**

A copy of a memo is kept in the [pack_nmemo](#) table if the memo is marked on an outstanding form.

Structure

The nmemo table has the following structure:

```
TABLE nmemo (
  node_id      INTEGER          NOT NULL,
  proc_id      INTEGER          NOT NULL,
  casenum      NUMERIC(20)      NOT NULL,
  memo_id      INTEGER          NOT NULL,
  memo_index   INTEGER          NOT NULL,
  memo_size    INTEGER          NOT NULL,
  memo_data    VARBINARY(MAX)   NOT NULL,
  array_idx    INTEGER          NOT NULL)
```

Column	Description
node_id	ID of the node that this memo is hosted on, as defined in the nodes table.
proc_id	ID of the procedure that this memo belongs to, as defined in the proc_index table.
casenum	Case number that this memo belongs to, as defined in the case_information table.
memo_id	Unique (for this case) ID of this memo.
memo_index	Index number into the set of rows that make up this memo. This value is always 1.
memo_size	Size (in bytes) of the memo data for this row.
memo_data	Memo data.
array_idx	Either: <ul style="list-style-type: none"> The array element index number of the memo.

Column	Description
	<ul style="list-style-type: none"> -1, if the memo is not an array memo field.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_nmemo	casenum memo_id memo_index proc_id node_id array_idx

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_nmemo_fk	casenum proc_id node_id

Table Activity

The nmemo table contains one row for each memo on the system. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	either: <ul style="list-style-type: none"> a memo field is first assigned. a memo field is modified.
updated	never.
deleted	either: <ul style="list-style-type: none"> a memo field is modified. a case is purged.

predict

The predict table stores the prediction data for all expected work items currently defined on the system.

Structure

The predict table has the following structure:

```
TABLE predict (
  node_id          INTEGER          NOT NULL,
  proc_num         NUMERIC(5)       NOT NULL,
  case_num         NUMERIC(20)      NOT NULL,
  parent_proc_num  NUMERIC(5)       NOT NULL,
  parent_case_num  NUMERIC(20)      NOT NULL,
  main_proc_num    NUMERIC(5)       NOT NULL,
  main_case_num    NUMERIC(20)      NOT NULL,
  step_name        VARCHAR(8)       NOT NULL,
  step_desc        VARCHAR(24)      NULL,
  step_desc2       VARCHAR(24)      NULL,
  step_addr        VARCHAR(49)      NOT NULL,
  step_durn_secs   NUMERIC(10)      NULL,
  step_durn_usecs  NUMERIC(10)      NULL,
  step_start       DATETIME         NOT NULL,
  step_start_usecs NUMERIC(10)      NOT NULL,
  step_end         DATETIME         NOT NULL,
  step_end_usecs   NUMERIC(10)      NOT NULL,
```

field_name	VARCHAR(31)	NULL,
field_value	VARCHAR(255)	NULL)

Column	Description
node_id	ID of the node that this predicted work item is hosted on, as defined in the nodes table.
proc_num	ID of the procedure associated with this predicted work item, as defined in the proc_index table.
case_num	Either: <ul style="list-style-type: none"> Case number of the case associated with this predicted work item, as defined in the case_information table. 0, if this is a predicted work item in a future sub-case, rather than in a currently outstanding sub-case.
parent_proc_num	ID of the parent procedure associated with this predicted work item, as defined in the proc_index table, if proc_num is a sub-procedure.
parent_case_num	Either: <ul style="list-style-type: none"> ID of the parent case associated with this predicted work item, as defined in the case_information table, if case_num is a sub-case. 0, if this is a predicted work item in a future sub-case, rather than a currently outstanding sub-case, that was itself started from a predicted future sub-case.
main_proc_num	ID of the procedure associated with the main case that generated this predicted work item, as defined in the proc_index table.
main_case_num	ID of the main case that generated this predicted work item, as defined in the case_information table.
step_name	Stepname of the step associated with this predicted work item.
step_desc	Step description of the step associated with this predicted work item.

Column	Description
step_desc2	Additional description of the step associated with this predicted work item.
step_addr	Queue name that this predicted work item will be delivered to.
step_durn_secs	Expected duration (in seconds) between this predicted work item being delivered to and released from the step_addr queue. Note: The step_durn_usecs column can be combined with this column to provide resolution to a microsecond.
step_durn_usecs	Number of microseconds to be added to the value specified in the step_durn_secs column.
step_start	Date and time that this predicted work item is expected to arrive in the step_addr queue, to the resolution of a second. Note: The step_start_usecs column can be combined with this column to provide resolution to a microsecond.
step_start_usecs	Number of microseconds since the start of the <i>seconds</i> value specified in the step_start column.
step_end	Date and time that this predicted work item is expected to be released from the step_addr queue, to the resolution of a second. Note: The step_end_usecs column can be combined with this column to provide resolution to a microsecond.
step_end_usecs	Number of microseconds since the start of the <i>seconds</i> value specified in the step_end column.
field_name	Name of the field that has a CDQP assigned to it for this predicted work item.
field_value	Value of the CDQP assigned to the field_name field for this predicted work item.

Primary Key

None.

Triggers

None.

Indexes

None.

Table Activity

The predict table contains one or more rows for each predicted work item generated by each step of each case of each procedure that currently has prediction data defined for it.

If a predicted work item contains one or more fields that have CDQPs assigned to them, duplicate rows are added for each CDQP. In the first row, the `field_name` and `field_value` columns are blank. Each subsequent row contains the `field_name` and `field_value` entries for one assigned CDQP. For example, if a predicted work item contains 5 fields that have CDQPs assigned to them, it will have 6 rows in this table.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	background prediction is enabled on the iProcess Engine, and anything occurs that causes prediction data for a case to be calculated or recalculated. For example, when a case is started, a work item is kept or released, a deadline expires or an event occurs. Note: One row is added for each step in the procedure that can occur on the currently predicted path(s).
updated	never.
deleted	background prediction is enabled on the iProcess Engine, and anything occurs that causes prediction data for a case to be recalculated. For example, when a work item is kept or released, a deadline expires or an event occurs.

A row is...	When...
-------------	---------

Note: All rows for a given main case number are deleted for each step in the procedure that can no longer occur on the currently predicted path(s).

**Note**

Case prediction can be enabled and disabled using the ENABLE_CASE_PREDICTION process attribute. See *TIBCO iProcess Engine Administrator's Guide* for more information.

predict_lock

The predict_lock table stores the locks that are used to control access to the predict table.

Structure

The predict_lock table has the following structure:

```
TABLE predict_lock (
  node_id          INTEGER          NOT NULL,
  proc_num         NUMERIC(5)       NOT NULL,
  case_num         NUMERIC(20)      NOT NULL)
```

Column	Description
node_id	ID of the node that this prediction lock is hosted on, as defined in the nodes table.
proc_num	ID of the procedure that this prediction lock applies to, as defined in the proc_index table.
case_num	Case number of the main case that this prediction lock applies to, as defined in the case_information table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_predict_lock	node_id proc_num case_num

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_predict_lock	DELETE	predict

Indexes

None.

Table Activity

The `predict_lock` table contains one row for every main case on the system that currently has prediction data defined in the [predict](#) table. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	background prediction is enabled on the iProcess Engine, and a case that has prediction enabled is started. Note: Case prediction can be enabled and disabled using the <code>ENABLE_CASE_PREDICTION</code> process attribute. See <i>TIBCO iProcess Engine Administrator's Guide</i> for more information.

A row is...	When...
updated	never.
deleted	a case that has prediction enabled is purged.

case_deadline_event

The `case_deadline_event` table stores information about case deadlines when the case is running.

Structure

The `case_deadline_event` table has the following structure:

```
TABLE case_deadline_event (
  node_id      INTEGER      NOT NULL,
  proc_id      INTEGER      NOT NULL,
  casenum      NUMERIC(20)   NOT NULL,
  dead_id      VARCHAR(32)   NOT NULL,
  dead_name    VARCHAR(32)   NOT NULL,
  event_name   VARCHAR(32)   NOT NULL,
  dead_value   VARCHAR(32)   NOT NULL)
```

Column	Description
<code>node_id</code>	ID of the node that this procedure is defined on, as defined in the nodes table.
<code>proc_id</code>	Unique ID of this procedure, generated from the sequences table.
<code>casenum</code>	The number of the case that this case deadline belongs to, as defined in the case_information table.
<code>dead_id</code>	For internal use only. This column is referenced from the stepname column in the outstanding_addr table.

Column	Description
dead_name	The name of the case deadline.
event_name	The name of the event step.
dead_value	<p>The value of the case deadline when the case is running. The value is specified in one of the following formats:</p> <ul style="list-style-type: none"> If the case deadline is specified as a period, then the value is in the format: <i>minutes^hours^days^weeks^months^years</i> If the case deadline is specified as an expression, then the value is in the format: <i>date expression^time expression</i>

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_case_dl_event	node_id proc_id casenum dead_id

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_case_dl_fk	casenum proc_id node_id

Table Activity

The `case_deadline_event` table contains one or more rows for each instance of each procedure definition on the system. Rows are added, updated, and deleted in the following situations.

A row is...	When...
added	If one of the following conditions occurs: <ul style="list-style-type: none"> • A case is starting and its deadline is defined in the procedure. • The <code>CreateCaseDeadline</code> expression is called in the EAI step.
updated	The <code>UpdateCaseDeadline</code> expression is called in the EAI step.
deleted	If one of the following conditions occurs: <ul style="list-style-type: none"> • The <code>DeleteCaseDeadline</code> expression is called in the EAI step. • The case deadline expired and an event is triggered. • The case is closed.

case_event

The `case_event` table stores information about cases that are interrupted by triggered events when processing the purge, close, resurrect, suspend, or resume operation. The case information is recorded in this table only when the BG process is handling the delayed release EAI steps, which are defined in the triggered event. After finishing the event, the case resumes execution and fetches the temporary case data from this table.

Structure

The case_event table has the following structure:

```
TABLE case_event(
    node_id          INTEGER          NOT NULL,
    proc_id          INTEGER          NOT NULL,
    major_vers       INTEGER          NOT NULL,
    minor_vers       INTEGER          NOT NULL,
    eventname        VARCHAR(32)      NOT NULL,
    user_event_name  VARCHAR(32)      NOT NULL,
    casenum          NUMERIC(20)      NOT NULL,
    state            INTEGER          NOT NULL,
    actionparameter  VARCHAR(256)     )
```

Column	Description
node_id	ID of the node that this case is hosted on, as defined in the nodes table.
proc_id	ID of the procedure that this procedure event belongs to, as defined in the proc_index table.
major_vers	Major version number of the procedure version that this case belongs to, as defined in the proc_version table.
minor_vers	Minor version number of the procedure version that this case belongs to, as defined in the proc_version table.
eventname	The name of the procedure event. The value of this column is one of the following: <ul style="list-style-type: none"> • BeforePurge • BeforeClose • AfterClose • BeforeResurrect • AfterResurrect • BeforeSuspend

Column	Description
	<ul style="list-style-type: none"> • AfterSuspend • BeforeResume • AfterResume
user_event_name	The name of the event step which you set for the procedure event.
casenum	ID of the case that this event belongs to, as defined in the case_information table.
state	<p>Flag that defines the state of the procedure event after the event is triggered. The meaning for each flag is:</p> <ul style="list-style-type: none"> • 2 the triggered event is in the processing state. • 3 the triggered event is finished. • 4 the triggered event is canceled. • -1 the triggered event failed.
actionparameter	<p>When an event is triggered, the processing purge, close, resurrect, suspend, or resume operation is interrupted. This column saves case data of the processing operation when the BG process is handling the delayed release EAI steps, which are defined in the triggered event. After finishing the event, the case resumes execution of the operation and fetches the temporary case data from this column.</p>

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_case_event	node_id proc_id casenum eventname

Triggers

None.

Index

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_case_event_fk	casenum proc_id node_id

Table Activity

The case_event table contains one or more rows for each instance of each procedure definition on the system. Rows are added, updated, and deleted in the following situations.

A row is...	When...
added	The procedure event enters the processing state.
updated	The procedure event changes from processing to failed or to canceled.
deleted	<p>If one of the following conditions occurred:</p> <ul style="list-style-type: none"> • The case is purged. • The procedure event is finished. • The procedure event failed. • The procedure event is canceled.

casenum_gaps

The casenum_gaps table holds the free case number gaps.

If the case number or the subcase number generated from the sequence table reaches the maximum case number, 4294967295, then the following cases cannot be started. This table is used to create more available case numbers by reusing previous blocks of case numbers, which are no longer exist. The free case numbers are available either because the case numbers have never been used or from the original cases that have been purged.

TIBCO iProcess Engine checks the `casenum_gaps` table to find out whether there are any free case numbers available for reuse before allocating a sequence from the end of the case numbers.

The `CASENUM_FIND_GAPS` stored procedure adds a list of free case number gaps to the `casenum_gaps` table, it scans a range of case numbers and create available blocks of free case numbers for reuse. See [CASENUM_FIND_GAPS](#) for more information.



Note

This table is not populated by the system and it remains empty unless the `CASENUM_FIND_GAPS` stored procedure is running to populate it.

Structure

The `casenum_gaps` table has the following structure:

```
TABLE casenum_gaps(
    casenum_min    bigint NOT NULL,
    casenum_max    bigint NOT NULL)
```

Column	Description
<code>casenum_min</code>	The minimum case number in a gap.
<code>casenum_max</code>	The maximum case number in a gap.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
<code>pk_casenum_gaps</code>	<code>casenum_min</code>

Foreign Keys

None.

Triggers

None.

Indexes

None.

Table Activity

The `casenum_gaps` table contains one or more rows for each instance of each procedure definition on the system. Rows are added, updated, and deleted in the following situations.

A row is...	When...
added	running the <code>CASENUM_FIND_GAPS</code> stored procedure.
updated	running TIBCO iProcess Engine.
deleted	running TIBCO iProcess Engine.

See Also

[CASENUM_FIND_GAPS](#)

Work Items

This section describes the tables that are used to store information about work item data - the combination of fields and their values that are held in iProcess forms (also known as “pack data”).

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



staffo

The **staffo** table holds information about *outstanding steps*, that is, steps that have been delivered to work queues but not yet released (or otherwise removed).

Structure

The staffo table is structured as follows:

```
TABLE staffo(
    o_flags            INTEGER            NULL,
    o_queueName        VARCHAR(24)        NULL,
    o_locker           VARCHAR(24)        NULL,
    o_username         VARCHAR(49)        NULL,
    o_startName        VARCHAR(49)        NULL,
    o_dirName          VARCHAR(12)        NOT NULL,
    o_dirDesc          VARCHAR(24)        NULL,
    o_procName         VARCHAR(8)         NOT NULL,
    o_procDesc         VARCHAR(24)        NULL,
    o_caseDesc         VARCHAR(24)        NULL,
    o_caseNum          NUMERIC(20)        NULL,
    o_placeno          INTEGER            NULL,
    o_dirFlags         INTEGER            NULL,
    o_procFlags        INTEGER            NULL,
    o_host             VARCHAR(24)        NOT NULL,
    o_pnum             INTEGER            NOT NULL,
    o_pnumcount        INTEGER            NOT NULL,
    o_casePtr          NUMERIC(20)        NULL,
    o_reqidHost        VARCHAR(24)        NOT NULL,
    o_reqid            NUMERIC(20)        NOT NULL,
    o_deadline         DATETIME           NULL,
    o_reqStamp         DATETIME           NULL,
    o_qparam1          VARCHAR(24)        NULL,
    o_qparam2          VARCHAR(24)        NULL,
    o_qparam3          VARCHAR(12)        NULL,
    o_qparam4          VARCHAR(12)        NULL,
    o_itemPriority      VARCHAR(24)        NULL,
    o_priorityChanged  DATETIME           NULL,
    o_majorVers        INTEGER            NOT NULL,
    o_minorVers        INTEGER            NOT NULL)
```

Column	Description
o_flags	Flags associated with this work item. For internal use only.
o_queueName	Queue name of the user or group queue that contains this work item, as defined in the user_names table.
o_locker	Name of the user who has locked the queue (if it is locked), as defined in

Column	Description
	the user_names table. Note: This column is not written to or updated unless the WIS_WRITELOCKS parameter in the <i>SWDIR\etc\staffcfg</i> file is set.
o_username	Queue name of the user or group queue that contains this work item, as defined in the user_names table.
o_startname	Username of the user who started the case that this work item belongs to, as defined in the user_names table.
o_dirname	Step name of the step that generated this work item.
o_dirdesc	Step description of the step that generated this work item.
o_procname	Procedure name of the procedure that generated this work item, as defined in the proc_index table.
o_procdesc	Procedure description of the procedure that generated this work item, as defined in the proc_index table.
o_casedesc	Case description of the case that this work item belongs to, as defined in the case_information table.
o_casenum	Case number of the case that this work item belongs to, as defined in the case_information table.
o_placeno	Step mark number. For internal use only.
o_dirflags	Step flags. For internal use only.
o_procflags	Procedure flags. For internal use only.
o_host	ID of the node that this work item is associated with, as defined in the nodes table.
o_pnum	Procedure number of the procedure that generated this work item, as

Column	Description
	defined in the proc_index table.
o_pnumcount	Version count of procedure. For internal use only.
o_caseptr	Case control record number. For internal use only.
o_reqidhost	Nodename of the node where the o_reqid is generated, as defined in the nodes table.
o_reqid	Unique ID for this work item, generated from the sequences table.
o_deadline	Date and time that the deadline (if defined) expires on this work item. If no deadline is set this value appears as 12/31/3000 11:15:00 PM.
o_reqstamp	Timestamp when this work item was delivered to the queue.
o_qparam1	Value of work queue parameter 1 for this work item.
o_qparam2	Value of work queue parameter 2 for this work item.
o_qparam3	Value of work queue parameter 3 for this work item.
o_qparam4	Value of work queue parameter 4 for this work item.
o_itempriority	<p>Priority definition for this work item, in the format:</p> <p><i>base:increment:number:period:type</i></p> <p>where:</p> <ul style="list-style-type: none"> • <i>base</i> is the base priority value for this work item. • <i>increment</i> is the amount that will be added to the item's priority value whenever the <i>period</i> expires. • <i>number</i> is the number of increments that will be added to the item's priority value. • <i>period</i> is the time period, in the units specified in <i>type</i>, which must

Column	Description
	<p>expire before the item's priority value is incremented.</p> <ul style="list-style-type: none"> <i>type</i> is the unit of measure of the <i>period</i>, either "M" or "m" for minutes, "H" or "h" for hours or "D" or "d" for days.
o_priority_changed	Timestamp when the priority value for this work item was last changed.
o_majorvers	Major version number of the procedure that generated this work item, as defined in the proc_version table.
o_minorvers	Minor version number of the procedure that generated this work item, as defined in the proc_version table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_staffo	o_reqid o_reqidhost

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_staffo	o_queuename

Index Name	Column(s) Indexed
idx_staffo_rowid ¹	rowid

Table Activity

The staffo table contains one row for every outstanding step on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a work item is sent out to a queue.
updated	any of the following occur: <ul style="list-style-type: none"> a work item is kept and any changes have been made. a work item's priority value changes. a work item is opened and the WIS_WRITELOCKS parameter in the <i>SWDIR\etc\staffcfg</i> file is set.
deleted	either: <ul style="list-style-type: none"> a work item is released or withdrawn. a case is closed or purged.

pack_data

The pack_data table holds the field name and value of every assigned field in every outstanding step on the system.

When a work item is sent out to a queue, field data is copied from the [case_data](#) table to the [pack_data](#) table. The client uses the field values in the [pack_data](#) table to fill out the form correctly. When the form is kept any changed fields are updated in the [pack_data](#)

¹UNIQUE index

table. When a work item is released field data is moved from the [pack_data](#) table to the [case_data](#) table.

Structure

The [pack_data](#) table has the following structure:

```
TABLE pack_data (
    reqid          NUMERIC(20)    NOT NULL,
    node_id        INTEGER        NOT NULL,
    proc_id        INTEGER        NOT NULL,
    casenum        NUMERIC(20)    NOT NULL,
    field_name     VARCHAR(31)    NOT NULL,
    field_value    VARCHAR(255)   NULL,
    field_flags    INTEGER        NOT NULL)
```

Column	Description
reqid	ID of the work item that this field belongs to, as defined in the staffo table.
node_id	ID of the node that this field is associated with, as defined in the nodes table.
proc_id	Number of the procedure that this field belongs to, as defined in the proc_index table.
casenum	Case number that this field belongs to, as defined in the case_information table.
field_name	Name of the field, as defined in the case_data table.
field_value	Value of the field. Note: A memo field has a value of 1. The associated memo data is stored in the pack_memo table.
field_flags	Status of the field. For internal use only.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_pack_data	reqid node_id field_name

Triggers

None.

Indexes

None.

Table Activity

The pack_data table contains one record for every assigned field that contains data (i.e. that has a value other than SW_NA) in every outstanding step on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Either: <ul style="list-style-type: none"> • a step is sent out. • a field is assigned a value on a keep or release.
updated	An assigned field has its value changed on a keep or release.
deleted	any of the following occur: <ul style="list-style-type: none"> • a release instruction for a work item is processed by the background process. • a work item is withdrawn.

A row is...	When...
-------------	---------

- | | |
|--|---|
| | <ul style="list-style-type: none"> a case is purged. |
|--|---|

pack_memo

The `pack_memo` table stores memo data associated with memo fields in the `pack_data` table before upgrading to iProcess Engine 11.6.

Since iProcess Engine version 11.6, all the new memo data associated with the memo fields is stored in the `VARBINARY(MAX)` data type in the `pack_nmemo` table.

11.6 You can migrate case memo data in the `pack_memo` table to the `pack_nmemo` table when upgrading to iProcess Engine or migrate them by using the `swutil MIGRATEMEMOS` command after the upgrades.

When a work item is sent out to a queue, memo data is copied from the `memo` table to the `pack_memo` table. The client uses the memo data in the `pack_memo` table to fill out the form correctly. When the form is kept any changed memo data is updated in the `pack_memo` table. When a work item is released memo data is moved from the `pack_memo` table to the `memo` table.

Structure

The `pack_memo` table is structured as follows:

```
TABLE pack_memo (
    reqid          NUMERIC(20)          NOT NULL,
    node_id        INTEGER              NOT NULL,
    proc_id        INTEGER              NOT NULL,
    casenum        NUMERIC(20)          NOT NULL,
    memo_id        INTEGER              NOT NULL,
    memo_index     INTEGER              NOT NULL,
    memo_size      INTEGER              NOT NULL,
    memo_data      VARBINARY(MAX)       NOT NULL,
    array_idx      INTEGER              NOT NULL)
```

Column	Description
reqid	ID of the work item that this memo belongs to, as defined in the staffo table.
node_id	ID of the node that this memo is associated with, as defined in the nodes table.
proc_id	Number of the procedure that this memo belongs to, as defined in the proc_index table.
casenum	Case number that this memo belongs to, as defined in the case_information table.
memo_id	Unique (for this case) ID of this memo.
memo_index	<p>Index number into the set of rows that make up this memo.</p> <p>If a memo is longer than 30,000 bytes multiple rows (in 30,000 byte chunks) are used to store the memo data. Each segment of the memo data is uniquely identified by its memo_index value.</p> <p>Note:</p> <ul style="list-style-type: none"> • If you have upgraded your system from a pre-Version i10.0-x(3.0) iProcess Engine, existing memos are divided into 2000 byte chunks. • However, if a memo is modified, the existing rows are deleted and then re-added using 30,000 byte chunks.
memo_size	Size (in bytes) of the memo data for this row.
memo_data	Memo data.
array_idx	<p>Either:</p> <ul style="list-style-type: none"> • The array element index number of the memo. • -1, if the memo is not an array memo field.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_pack_memo	reqid node_id casenum memo_id memo_index array_idx

Triggers

None.

Indexes

None.

Table Activity

The pack_memo table contains one or more rows for every assigned memo field that contains data (i.e. that has a value other than SW_NA) in every outstanding step on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Either: <ul style="list-style-type: none"> • a step containing memo data is sent out. • a memo field is assigned a value on a keep or release.
updated	An assigned memo field has its data changed on a keep or release.
deleted	any of the following occur: <ul style="list-style-type: none"> • a release instruction for a work item containing memo data is processed by the background process. • a work item containing memo data is withdrawn. • a case containing memo data is purged.

pack_nmemo

The pack_nmemo table stores memo data associated with memo fields in the [pack_data](#) table after upgrading to iProcess Engine 11.6.

Since iProcess Engine version 11.6, all the new memo data associated with the memo fields is stored in the VARBINARY(MAX) data type in the pack_nmemo table.

You can migrate case memo data in the [pack_memo](#) table to the pack_nmemo table when upgrading to iProcess Engine 11.6 or migrate them by using the `swutil MIGRATEMEMOS` command after the upgrades.

When a work item is sent out to a queue, memo data is copied from the [nmemo](#) table to the [pack_nmemo](#) table. The client uses the memo data in the [pack_nmemo](#) table to fill out the form correctly. When the form is kept any changed memo data is updated in the [pack_nmemo](#) table. When a work item is released memo data is moved from the [pack_nmemo](#) table to the [nmemo](#) table.

Structure

The pack_nmemo table is structured as follows:

```
TABLE pack_nmemo (
    reqid          NUMERIC(20)          NOT NULL,
    node_id        INTEGER              NOT NULL,
    proc_id        INTEGER              NOT NULL,
    casenum        NUMERIC(20)          NOT NULL,
    memo_id        INTEGER              NOT NULL,
    memo_index     INTEGER              NOT NULL,
    memo_size      INTEGER              NOT NULL,
    memo_data      VARBINARY(MAX)       NOT NULL,
    array_idx      INTEGER              NOT NULL)
```

Column	Description
reqid	ID of the work item that this memo belongs to, as defined in the staffo table.
node_id	ID of the node that this memo is associated with, as defined in the nodes table.
proc_id	Number of the procedure that this memo belongs to, as defined in the proc_

Column	Description
	index table.
casenum	Case number that this memo belongs to, as defined in the case_information table.
memo_id	Unique (for this case) ID of this memo.
memo_index	Index number into the set of rows that make up this memo. This value is always 1.
memo_size	Size (in bytes) of the memo data for this row.
memo_data	Memo data.
array_idx	Either: <ul style="list-style-type: none"> The array element index number of the memo. -1, if the memo is not an array memo field.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_pack_nmemo	reqid node_id casenum memo_id memo_index array_idx

Triggers

None.

Indexes

None.

Table Activity

The pack_nmemo table contains one row for every assigned memo field that contains data (for example, that has a value other than SW_NA) in every outstanding step on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Either: <ul style="list-style-type: none"> • a step containing memo data is sent out. • a memo field is assigned a value on a keep or release.
updated	An assigned memo field has its data changed on a keep or release.
deleted	any of the following occur: <ul style="list-style-type: none"> • a release instruction for a work item containing memo data is processed by the background process. • a work item containing memo data is withdrawn. • a case containing memo data is purged.

qaccess

The qaccess table stores details of any non-default sort, filter and display criteria used by iProcess users to access their iProcess queues.

Structure

The qaccess table has the following structure:

```
TABLE qaccess (
    user_name          VARCHAR(64)    NOT NULL,
```

access_type	VARCHAR(8)	NOT NULL,
queue_name	VARCHAR(51)	NOT NULL,
access_str	VARCHAR(1024)	NULL)

Column	Description
user_name	Name of the user that this row applies to, as defined in the user_names table.
access_type	Type of access criteria defined in this row. Any of the following: <ul style="list-style-type: none"> • SORT defines how work items in the specified queue are sorted. • FILTER defines how work items in the specified queue are filtered. • DISPLAY defines how work items in the specified queue are displayed. • QVERS defines when the queue was last accessed. For internal use only.
queue_name	Name of the (user or group) queue that this row applies to, as defined in the proc_version table. Note: Test queues have the suffix /t.
access_str	Access criteria. For internal use only.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_qaccess	user_name queue_name access_type

Triggers

None.

Indexes

None.

Table Activity

The `qaccess` table contains one row per set of non-default access criteria defined per user per queue.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a set of non-default access criteria is saved for a user.
updated	a set of non-default access criteria is updated for a user.
deleted	a user reverts to using the default criteria.

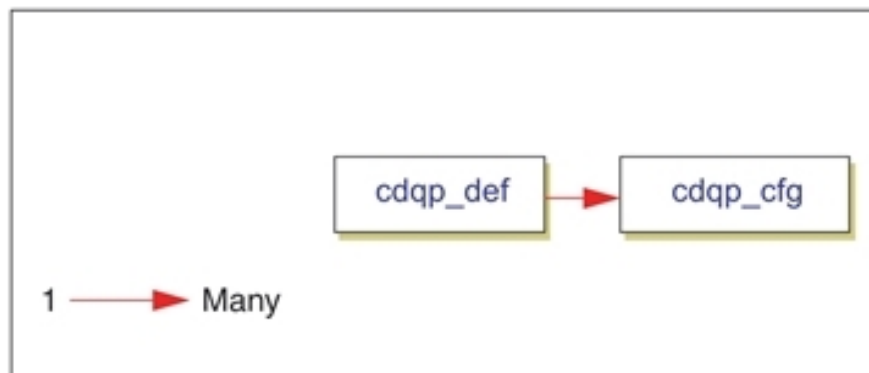
Case Data Queue Parameters

This section describes the tables that are used to store information about Case Data Queue Parameters (CDQPs).

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



cdqp_def

The `cdqp_def` table holds information about each field that is defined as a Case Data Queue Parameter (CDQP).

Structure

The `cdqp_def` table has the following structure:

```
TABLE cdqp_def (
    def_id          NUMERIC(10)      NOT NULL,
    field_name      VARCHAR(31)      NOT NULL,
    data_size       NUMERIC(5)       NOT NULL,
    description     VARCHAR(40)      NOT NULL,
    is_predict      SMALLINT         NOT NULL)
```

Column	Description
def_id	Unique identifier for this CDQP, generated from the sequences table.
field_name	Name of the iProcess field assigned to this CDQP, as defined in the case_data table.
data_size	Maximum size, in characters, of this CDQP.
description	Name used to represent this CDQP in Work Queue Manager dialogs.
is_predict	Flag that defines whether (1) or not (0) this CDQP is used for case prediction.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_cdqp_def	def_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_cdqp_def	DELETE	cdqp_cfg

Indexes

None.

Table Activity

The `cdqp_def` table contains one row for each field on the system that is currently defined as a CDQP.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a field is first defined as a CDQP.
updated	an existing CDQP definition is updated.
deleted	an existing CDQP definition is deleted.

cdqp_cfg

The `cdqp_cfg` table holds the details of each mapping of a CDQP to a queue.

Structure

The `cdqp_cfg` table has the following structure:

```
TABLE cdqp_cfg (
  cfg_id          NUMERIC(10)    NOT NULL,
  def_id          NUMERIC(10)    NOT NULL,
  queue_name      VARCHAR(48)    NOT NULL)
```

Column	Description
<code>cfg_id</code>	Unique ID for this CDQP/queue mapping generated from the sequences table.

Column	Description
def_id	ID of the CDQP that is mapped to the queue_name queue, as defined in the cdqp_def table.
queue_name	Name of the iProcess queue that the CDQP defined in def_id is mapped to, as defined in the user_names table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_cdqp_cfg	cfg_id

Triggers

None.

Indexes

None.

Table Activity

The cdqp_cfg table contains one row for each mapping of a CDQP to a queue that is defined on the system. For example, if CDQP1 is mapped to 6 queues, and CDQP2 is mapped to 4 queues, the cdqp_cfg table contains 10 rows.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a field (that is already defined as a CDQP) is mapped to a queue.

A row is...	When...
updated	never.
deleted	an existing CDQP mapping is deleted.

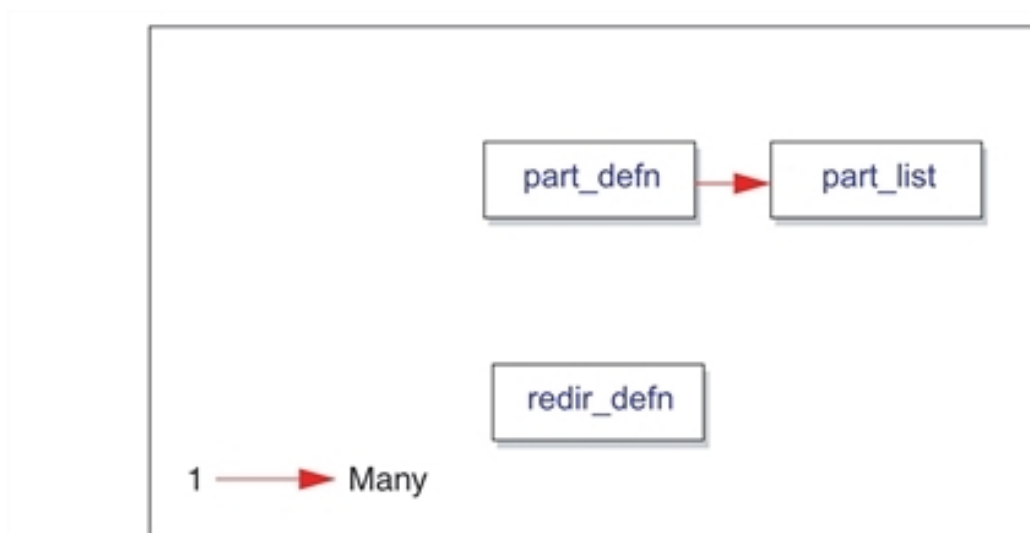
Queue Participation and Redirection

This section describes the tables that are used to store information about iProcess participation and redirection records.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



part_defn

The `part_defn` table holds all the *participation records* defined on the system. A participation record defines the dates and times that users are allowed to participate in a particular queue. (The [part_list](#) table defines what users are allowed to use a particular participation record.)

Structure

The part_defn table has the following structure:

```
TABLE part_defn (
    part_id          INTEGER          NOT NULL,
    queue_name       VARCHAR(24)      NOT NULL,
    days_mask        VARCHAR(7)       NOT NULL,
    start_time       SMALLINT         NOT NULL,
    end_time         SMALLINT         NOT NULL,
    style            VARCHAR(24)      NULL,
    start_date       INTEGER          NOT NULL,
    end_date         INTEGER          NOT NULL)
```

Column	Description
part_id	Unique ID for this participation record.
queue_name	Name of the queue that this participation record allows users to participate in, as defined in the user_names table.
days_mask	Days of the week that users can participate in the specified queue_name. For example, -TWT-SS indicates every day except Monday or Friday.
start_time	Time of day when participation starts.
end_time	Time of day when participation ends.
style	<i>Not used. Reserved for possible future use.</i>
start_date	Date on which participation starts.
end_date	Date on which participation ends.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_part_defn	part_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_part_defn	DELETE	part_list

Indexes

None.

Table Activity

The part_defn table contains one row for each participation record defined on the system. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new participation record is added.
updated	an existing participation record is updated.
deleted	an existing participation record is deleted.

part_list

The part_list table holds the names of all users who are currently allowed to participate in other queues.

Structure

The part_list table has the following structure:

```
TABLE part_list (
    part_id          INTEGER          NOT NULL,
    user_name        VARCHAR(64)     NOT NULL)
```

Column	Description
part_id	ID of the participation record that this participant is a member of, as defined in the part_defn table.
user_name	Name of the user who is allowed to participate (according to the participation definition identified by the part_id value), as defined in the user_names table.

Primary Key

None.

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) indexed
idx_part_list_fk	part_id

Table Activity

The part_list table contains one record for each user designated as a participant in each participation record on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Either: <ul style="list-style-type: none"> a new participation record is added. an existing participation record is updated (if a user is added as part of the update).
updated	never.
deleted	Either: <ul style="list-style-type: none"> an existing participation record is deleted. an existing participation record is updated (if a user is deleted as part of the update).

redir_defn

The `redir_defn` table holds information about which queues are being redirected and which queues they are being redirected to.

Structure

The `redir_defn` table has the following structure:

```
TABLE redir_defn (
    redir_id          INTEGER          NOT NULL,
    start_time        SMALLINT         NOT NULL,
    start_date        INTEGER          NOT NULL,
    end_time          SMALLINT         NOT NULL,
    end_date          INTEGER          NOT NULL,
    queue_name        VARCHAR(24)      NOT NULL,
    destination       VARCHAR(49)      NOT NULL)
```

Column	Description
redir_id	Unique ID for this redirection record.
start_time	Time that this queue redirection starts.
start_date	Date that this queue redirection starts.
end_time	Time that this queue redirection ends.
end_date	Date that this queue redirection ends.
queue_name	Name of the queue from which work items are to be redirected, as defined in the user_names table.
destination	Name of the queue to which work items are to be redirected, as defined in the user_names table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_redir_defn	redir_id

Triggers

None.

Indexes

None.

Table Activity

The `redir_defn` table contains one record for each redirection record defined on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a queue is redirected.
updated	the details of an existing redirection are updated.
deleted	redirection for a queue is turned off.

Administrative Tables

This section describes the tables that are used to store administrative information about the iProcess system.

Table Relationships

The [flag_table](#) and [version](#) tables have no trigger-enforced relationships with each other or with any other tables.

flag_table

The `flag_table` table provides a locking mechanism which controls access to the four areas of iProcess administrative data - users, lists, roles and TIBCO iProcess® Engine tables. iProcess administrative data is maintained in two sets of tables:

- The main system data, which iProcess references during normal operation, is stored in tables without a prefix (for example, [user_names](#) or [dbs_fields](#)).
- A copy of this data, containing users' edits that have not yet been released for use by the system, is stored in identical tables which have the same name prefixed by `tsys_` (for example, [tsys_user_names](#) or [tsys_dbs_fields](#)).

The `flag_table` table contains a row for each area of iProcess administrative data, and is used to prevent multiple users from editing the same data at the same time.

When a user edits the data in a particular row (for example, using User Manager to edit user data), the `area_locked` flag is set while editing takes place. On completion of the edit, the `area_locked` flag is cleared. If changes have been made, the `area_changed` flag is set.

When a user requests a Move System Information, the `move_req` flag is set on any rows that have the `area_changed` flag set. When the background process sees a row with `move_req` flagged that is not locked, it locks the area and updates the main system data tables from the `tsys_` tables. When the Move System Information operation completes, all the flags are cleared.

Structure

The flag_table table has the following structure:

```
TABLE flag_table (
    area_id          INTEGER          NOT NULL,
    area_locked      INTEGER          NOT NULL,
    area_changed     INTEGER          NOT NULL,
    move_req         INTEGER          NOT NULL,
    user_name        VARCHAR(64)      NULL)
```

Column	Description
area_id	Unique ID of this area of iProcess administrative data: Either Users (1), iProcess Tables (2), Lists (3) or Roles (4).
area_locked	Flag that defines whether (1) or not (0) the specified area_id is locked. The flag is set by: <ul style="list-style-type: none"> an editor (for example, User Manager) when a user is editing the specified area, to prevent other users from editing the same data. the background process while it is updating the system data, to prevent any users from editing the same data.
area_changed	Flag that defines whether (1) or not (0) the tsys_ tables for the specified area_id contain modified data.
move_req	Flag that defines whether (1) or not (0) the specified area_id needs to be updated by a Move System Information operation.
user_name	Name of the user currently altering data in the given area, as defined in the user_names table. This is either: <ul style="list-style-type: none"> the name of the user doing the editing, or swpro if the background process has the area locked.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_flag_table	area_id

Triggers

None.

Indexes

None.

Table Activity

The flag_table table always contains 4 rows - one row for each area of iProcess administrative data (users, lists, roles and TIBCO iProcess® Engine tables).

The table is populated when the iProcess Engine is installed.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	never.
updated	either: <ul style="list-style-type: none"> a Move System Information operation either starts or completes. an edit of a data area either starts or completes.
deleted	never.

version

The version table contains version information on system data: currently either CDQP or user data. Processes that hold user details query this table to determine if their internal cache is up to date or not.

Structure

The version table has the following structure:

```
TABLE version (  
    version_type    VARCHAR(20)    NOT NULL,  
    version_value   INTEGER        NOT NULL)
```

Column	Description
version_type	Data type: either cdqp or user.
version_value	Number that is incremented whenever the data is changed.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_version	version_type

Triggers

None.

Indexes

None.

Table Activity

The version table always contains a single row. The table is populated when the iProcess Engine is installed.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	never.
updated	a Move System Information operation is performed and data for users, groups or attributes has been modified (that is, if the move_req flag for the Users data area in the flag_table is set to 1).
deleted	never.

Users and Work Queues

This section describes the tables that are used to store information about iProcess user and group queues.

About User Tables

Note that there are two sets of user tables:

- The tables prefixed with `user_` hold the main system data, which TIBCO iProcess® Engine references during normal operation.
- The tables prefixed with `tsys_user_` hold a copy of this data, containing users' edits that have not yet been released for use by the system.

The `tsys_user_` tables are purged and rewritten whenever a user edits user data (either by saving changes made in the User Manager utility in the TIBCO iProcess Administrator, importing data with `SWDIR\bin\swutil USERINFO`, or by using TIBCO iProcess Objects).

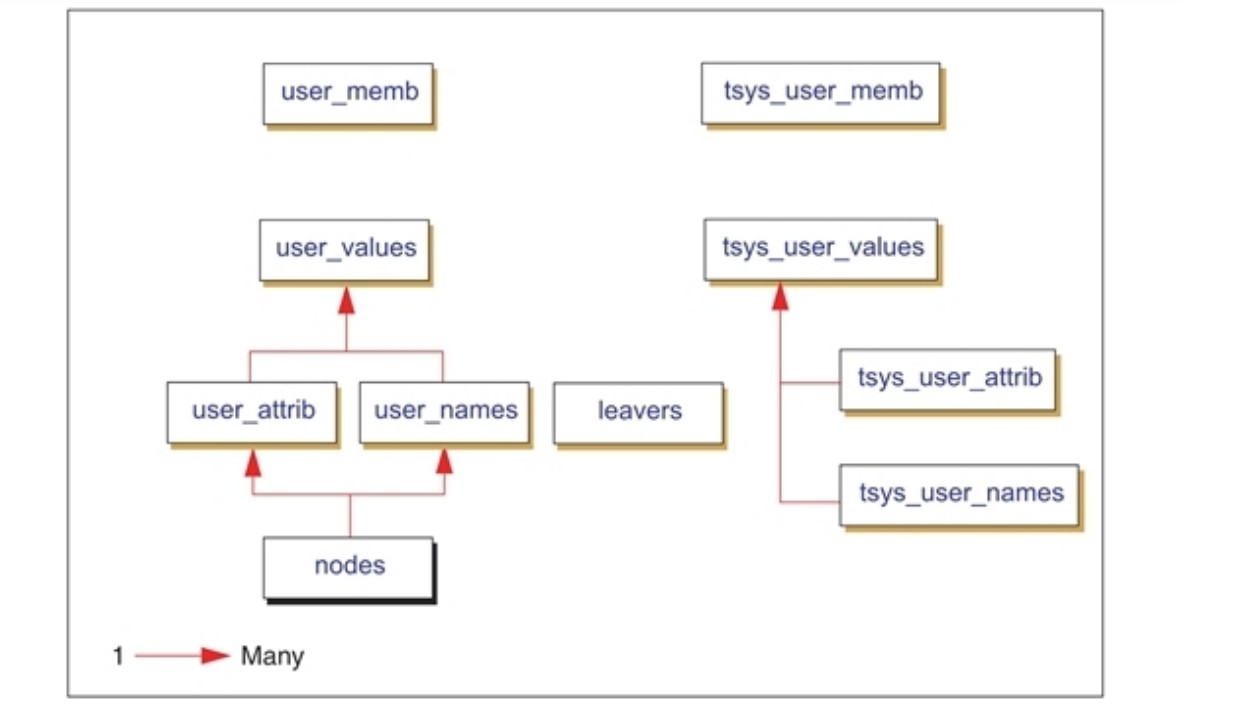
The `user_` tables are purged and rewritten with the updated information from the `tsys_user_` tables when a Move System Information is performed - *if* the [flag_table](#) indicates that the appropriate data area has been modified.

Access to the `user_` and `tsys_user_` tables is controlled by a locking mechanism provided by the [flag_table](#) table.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



user_names

The user_names table holds the names of all iProcess user and group queues registered on the system.

Structure

The user_names table has the following structure:

```
TABLE user_names (  
    node_id      INTEGER      NOT NULL,  
    user_id      INTEGER      NOT NULL,  
    user_name     VARCHAR(64)  NOT NULL,  
    user_type     VARCHAR(1)   NOT NULL)
```

Column	Description
node_id	ID of the node that this (user or group) queue is registered on, as defined in the

Column	Description
	nodes table.
user_id	Unique ID for this (user or group) queue. Note: Users and groups have separate ID sequences, as defined in the user_type column, so both a user and a group can have the same user_id value.
user_name	Name of this (user or group) queue.
user_type	Queue type: user (U) or group (G).

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_user_names	user_id user_type node_id

Triggers

The following DELETE CASCADE triggers are defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_user_names	DELETE	user_values
tr_user_memb_del	DELETE	user_memb

Indexes

The following indexes are defined for this table.

Index Name	Column(s) Indexed
idx_user_names_fk	node_id
idx_user_names	user_name

Table Activity

The user_names table contains one row for each user or group queue defined on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	<p>a Move System Information is performed, if the flag_table indicates that the Users data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_user_names table.</p>
updated	never.
deleted	<p>a Move System Information is performed, if the flag_table indicates that the Users data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_user_names table.</p>

user_attrib

The user_attrib table holds the definitions of all iProcess attributes defined on the system.

Structure

The user_attrib table has the following structure:

```
TABLE user_attrib (
    node_id      INTEGER      NOT NULL,
    attribute_id  INTEGER      NOT NULL,
    attribute_name VARCHAR(15) NOT NULL,
    attribute_type VARCHAR(1)  NOT NULL)
```

Column	Description
node_id	ID of the node that this attribute is defined on, as defined in the nodes table.
attribute_id	Unique ID for this attribute.
attribute_name	Name of this attribute.
attribute_type	Attribute type: Either ASCII (A), Numeric (R), Date (D) or Time (T).

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_user_attrib	attribute_id node_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_user_attrib	DELETE	user_values

Indexes

The following indexes are defined for this table.

Index Name	Column(s) Indexed
idx_user_attrib_fk	node_id
idx_user_attrib	attribute_name

Table Activity

The user_attrib table contains one or more rows for each iProcess attribute defined on the system. If an attribute's maximum length is defined as:

- 24 characters or less, one row is created for the attribute.
- 25 characters or more, one row is created for each 24 characters of the attribute's maximum length, and a number is appended to the attribute_name entry for each row.

The following example illustrates this:

- DESCRIPTION is a system-defined attribute of type ASCII with a maximum length of 24 characters; one row is therefore added to the table.
- QSUPERVISORS is a system-defined attribute of type ASCII with a maximum length of 48 characters; two rows are therefore added to the table - QSUPERVISORS_01 and QSUPERVISORS_02, each with a unique attribute_id.
- JOBDESC is a user-defined attribute of type ASCII with a maximum length of 60 characters; two rows are therefore added to the table - JOBDESC_01 and JOBDESC_02, each with a unique attribute_id.

node_id	attribute_id	attribute_name	attribute_type
1	1	DESCRIPTION	A
1	2	LANGUAGE	A
1	3	MENUNAME	A
1	4	SORTMAIL	A

1	5	USERFLAGS	A
1	6	QSUPERVISORS_01	A
1	7	QSUPERVISORS_02	A
1	9	JOBDESC_01	A
1	10	JOBDESC_02	A

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	<p>a Move System Information is performed, if the flag_table indicates that the Users data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_user_names table.</p>
updated	never.
deleted	<p>a Move System Information is performed, if the flag_table indicates that the Users data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_user_names table.</p>

user_setting

The user_setting table holds the settings that a given user has defined in the iProcess Workspace (Browser). This enables a user to keep the same settings when working on any machine.

Structure

The user_setting table has the following structure:

```
TABLE user_setting (
  username      varchar(32)      NOT NULL,
  userkey       varchar(128)     NOT NULL
  valindex      integer          NOT NULL,
```

vallen	integer	NOT NULL
uservalue	varbinary(max)	NULL

Column	Description
username	The name of the user whose preferences these are, as defined in the user_names table.
userkey	The key of the user.
valindex	<p>The index number into the set of rows that make up the user value.</p> <p>If the user value is longer than 30,000 bytes, multiple rows (in 30,000 byte chunks) are used to store the user value. Each segment of the user value is uniquely identified by its valindex value.</p> <p>Note: Since iProcess Engine 11.6, the volume of a row is upgraded. The value of valindex might always be 1.</p>
vallen	The size (in bytes) of the memo data for this row.
uservalue	The value of the particular user identified by userkey.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_usersetting	username userkey valindex

user_values

The **user_values** table holds the values for all attributes defined for all users and groups on the system.

Structure

The user_values table has the following structure:

```
TABLE user_values (
    node_id          INTEGER          NOT NULL,
    user_id          INTEGER          NOT NULL,
    attribute_id     INTEGER          NOT NULL,
    attribute_value   VARCHAR(24)     NULL,
    user_type        VARCHAR(1)       NOT NULL)
```

Column	Description
node_id	ID of the node that this attribute is defined on, as defined in the nodes table.
user_id	ID of the (user or group) queue that this attribute value is associated with, as defined in the user_names table.
attribute_id	ID of the attribute that this attribute value is associated with, as defined in the user_attrib table.
attribute_value	Value of this attribute. Note: If an attribute value is longer than 24 characters multiple rows are used to store the value. Each segment of the value is uniquely identified by its attribute_id value, as defined in the user_attrib table.
user_type	Type of the (user or group) queue that this attribute value is associated with, as defined in the user_names table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_user_values	user_id attribute_id user_type node_id

Triggers

None.

Indexes

The following indexes are defined for this table.

Index Name	Column(s) Indexed
idx_user_values_fk1	node_id user_id user_type
idx_user_values_fk2	attribute_id node_id
idx_user_values	attribute_id

Table Activity

The user_values table contains one or more rows per assigned attribute per (user or group) queue on the system. If an attribute value's length is:

- 24 characters or less, one row is created for the attribute value.
- more than 24 characters, one row is created for each 24 characters of the attribute value.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	<p>a Move System Information is performed, if the flag_table indicates that the Users data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_user_names table.</p>
updated	never.

A row is...	When...
deleted	a Move System Information is performed, if the flag_table indicates that the Users data area has been modified.
	Note: The table is purged and rewritten using the values from the tsys_user_names table.

user_memb

The user_memb table defines users' membership of groups.

Structure

The user_memb table has the following structure:

```
TABLE user_memb (
    node_id      INTEGER      NOT NULL,
    user_id      INTEGER      NOT NULL,
    group_id     INTEGER      NOT NULL)
```

Column	Description
node_id	ID of the node that this user/group combination is defined on, as defined in the nodes table.
user_id	ID of the user who belongs to the group, as defined in the user_names table.
group_id	ID of the group that the user belongs to, as defined in the user_names table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_user_memb	user_id node_id group_id

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_user_memb	user_id group_id

Table Activity

The user_memb table contains one row for every user/group member relationship defined on the system. For example, if a user is a member of three different groups, there are three rows for that user in this table.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	<p>a Move System Information is performed, if the flag_table indicates that the Users data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_user_names table.</p>
updated	never.
deleted	a Move System Information is performed, if the flag_table indicates that the

A row is...	When...
-------------	---------

	Users data area has been modified.
--	------------------------------------

Note: The table is purged and rewritten using the values from the [tsys_user_names](#) table.

leavers

The leavers table stores information about the recently deleted users.

Structure

The leavers table has the following structure:

```
TABLE leavers (
  node_id          INTEGER          NOT NULL,
  user_name        VARCHAR(64)      NOT NULL,
  destination      VARCHAR(64)      NOT NULL,
  timestamp        NUMERIC(20)      NOT NULL,
  status           INTEGER          NOT NULL)
```

Column	Description
node_id	ID of the node that this (user or group) queue is registered on, as defined in the nodes table.
user_name	Name of this deleted user.
destination	Description of this deleted user.
timestamp	When the current status is set.
status	Status of the redirection performed on the leaver. One of the following values: <ul style="list-style-type: none"> 0 (LEAVER_WILL_BE_REDIRECTED) The leaver will be redirected.

Column	Description
	<ul style="list-style-type: none"> • 1 (LEAVER_IS_BEING_REDIRECTED) The leaver is being redirected. • 2 (LEAVER_FINISH_REDIRECTION) The leaver has been redirected.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_leavers	user_name node_id

Triggers

None.

Indexes

None.

Table Activity

The leavers table contains one row for each recently deleted user.

Rows are added, updated, and deleted in the following situations.

A row is...	When...
added	a user is deleted.
updated	one of the following conditions is met: <ul style="list-style-type: none"> • the iProcess Engine is started, • the status of the deleted user is changed.

A row is...	When...
deleted	<p>all of the following conditions are met:</p> <ul style="list-style-type: none"> the status field is set to 2 (LEAVER_FINISH_REDIRECTION), the time length defined by the WQS_LEAVER_PERIOD process attribute has passed since the status field was set to 2, the iProcess Engine is shut down, or a Move System Information operation is performed.

tsys_user_names

The tsys_user_names table is a copy of the [user_names](#) table. It is identical to the [user_names](#) table except for the following:

- The primary key name is pk_tsys_user_names.
- No indexes are defined.
- The following DELETE CASCADE triggers are defined.

Trigger Name	Triggered by	Affects Table(s)
tr_tsys_user_names	DELETE	tsys_user_values
tr_tsys_user_memdb	DELETE	tsys_user_memb

- The table is purged and rewritten when a user edits user data, either by saving changes made in the User Manager utility in the TIBCO iProcess Administrator, importing data with *SWDIR\bin\swutil USERINFO*, or by using TIBCO iProcess Objects. (The [flag_table](#) is also updated to indicate that the Users data area has been modified.)

tsys_user_attrb

The tsys_user_attrb table is a copy of the [user_attrb](#) table. It is identical to the [user_attrb](#) table except for the following:

- The primary key name is pk_tsys_user_attrb.
- No indexes are defined.
- The following DELETE CASCADE trigger is defined.

Trigger Name	Triggered by	Affects Table(s)
tr_tsys_user_attrb	DELETE	tsys_user_values

- The table is purged and rewritten when a user edits user data, either by saving changes made in the User Manager utility in the TIBCO iProcess Administrator, importing data with *SWDIR\bin\swutil USERINFO*, or by using TIBCO iProcess Objects. (The [flag_table](#) is also updated to indicate that the Users data area has been modified.)

tsys_user_values

The tsys_user_values table is a copy of the [user_values](#) table. It is identical to the [user_values](#) table except for the following:

- The primary key name is pk_tsys_user_values.
- The index names are idx_tsys_user_values_fk1 and idx_tsys_user_values_fk2.
- The table is purged and rewritten when a user edits user data, either by saving changes made in the User Manager utility in the TIBCO iProcess Administrator, importing data with *SWDIR\bin\swutil USERINFO*, or by using TIBCO iProcess Objects. (The [flag_table](#) is also updated to indicate that the Users data area has been modified.)

tsys_user_memb

The `tsys_user_memb` table is a copy of the `user_memb` table. It is identical to the `user_memb` table except for the following:

- The primary key name is `pk_tsys_user_memb`.
- The index name is `idx_tsys_user_memb`.
- The table is purged and rewritten when a user edits user data, either by saving changes made in the User Manager utility in the TIBCO iProcess Administrator, importing data with `SWDIR\bin\swutil USERINFO`, or by using TIBCO iProcess Objects. (The `flag_table` is also updated to indicate that the Users data area has been modified.)

Roles

This section describes the tables that are used to store information about iProcess roles.

About Roles

Note that:

- The [role_users](#) table holds the main system data, which TIBCO iProcess® Engine references during normal operation.
- The [tsys_role_users](#) holds a copy of this data, containing users' edits that have not yet been released for use by the system.

The [tsys_role_users](#) table is purged and rewritten whenever a user edits role data (either by saving changes made in the User Manager utility in the TIBCO iProcess Administrator, importing data with `SWDIR\bin\swutil ROLEINFO`, or by using TIBCO iProcess Objects).

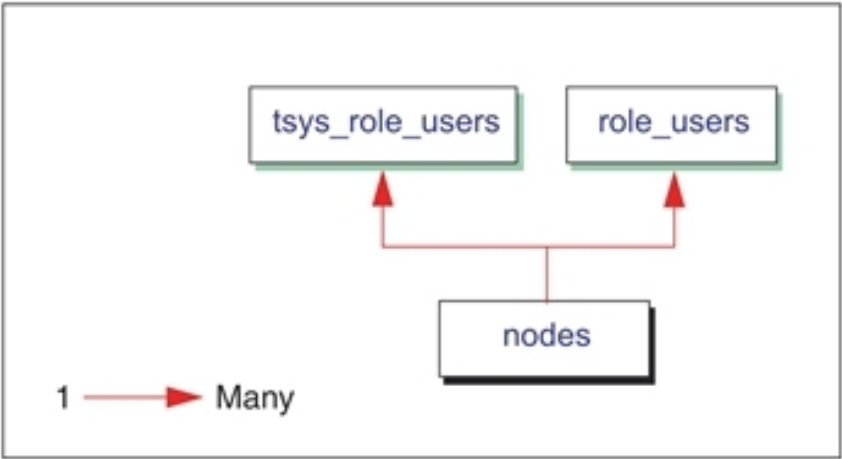
When a Move System Information is performed, if the [tsys_role_users](#) table has been changed, the [role_users](#) table is purged and rewritten with the updated information from the [tsys_role_users](#) table.

Access to the [role_users](#) and [tsys_role_users](#) tables is controlled by a locking mechanism provided by the [flag_table](#) table.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



role_users

The `role_users` table holds information about which roles are defined on the system, and which queues are assigned to them.

Structure

The `role_users` table has the following structure:

```
TABLE role_users (  
    node_id      INTEGER          NOT NULL,  
    role_id      INTEGER          NOT NULL,  
    role_name    VARCHAR(15)      NOT NULL,  
    usernode_name VARCHAR(49)      NOT NULL)
```

Column	Description
node_id	ID of the node that this role is registered on, as defined in the nodes table.
role_id	Unique ID for this role.
role_name	Name of this role.
usernode_name	Name of the (user or group) queue that the role is assigned to, as defined in the user_names table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_role_users	role_id node_id

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_role_users_fk	node_id

Table Activity

The `role_users` table contains one row for each role defined on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	<p>a Move System Information is performed, if the flag_table indicates that the Roles data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_role_users table.</p>
updated	never.
deleted	a Move System Information is performed, if the flag_table indicates that the

A row is...	When...
	Roles data area has been modified.
	Note: The table is purged and rewritten using the values from the tsys_role_users table.

tsys_role_users

The `tsys_role_users` table is a copy of the [role_users](#) table. It is identical to the [role_users](#) table except for the following:

- The primary key name is `pk_tsys_role_users`.
- The index name is `idx_tsys_role_users_fk`.
- The table is purged and rewritten when a user edits role data, either by saving changes made in the User Manager utility in the TIBCO iProcess Administrator, importing data with `SWDIR\bin\swutil ROLEINFO`, or by using TIBCO iProcess Objects. (The [flag_table](#) is also updated to indicate that the Roles data area has been modified.)

TIBCO iProcess Tables

This section describes the tables that are used to store information about TIBCO iProcess tables.



Note

This section uses the term *TIBCO iProcess table* to mean an iProcess table, and *table* to mean a SQL Server table.

About TIBCO iProcess Tables

Note that there are four sets of related tables, as follows:

Prefix	Description
db_	Hold the main system data on <i>installed</i> TIBCO iProcess tables, which iProcess references during normal operation.
str_ db_	Hold the main system data on <i>uninstalled</i> TIBCO iProcess tables, which iProcess references during normal operation. Note: There is no str_db_values table, because no data is associated with uninstalled TIBCO iProcess tables.
tsys_ db_	Hold a copy of the main (db_ and str_db_) system data, containing users' edits that have not yet been released for use by the system.
tmp_ db_	Temporary tables used only when importing TIBCO iProcess® Engine tables (using <code>SWDIR\bin\swutil IMPORT</code>).

The tsys_db_ tables are purged and rewritten whenever a user edits TIBCO iProcess® Engine table data (either by saving changes made in the Table Manager utility in the TIBCO iProcess Administrator, modifying data with `SWDIR\bin\swutil IMPORT` or `DELTAB`, or by using TIBCO iProcess Objects).

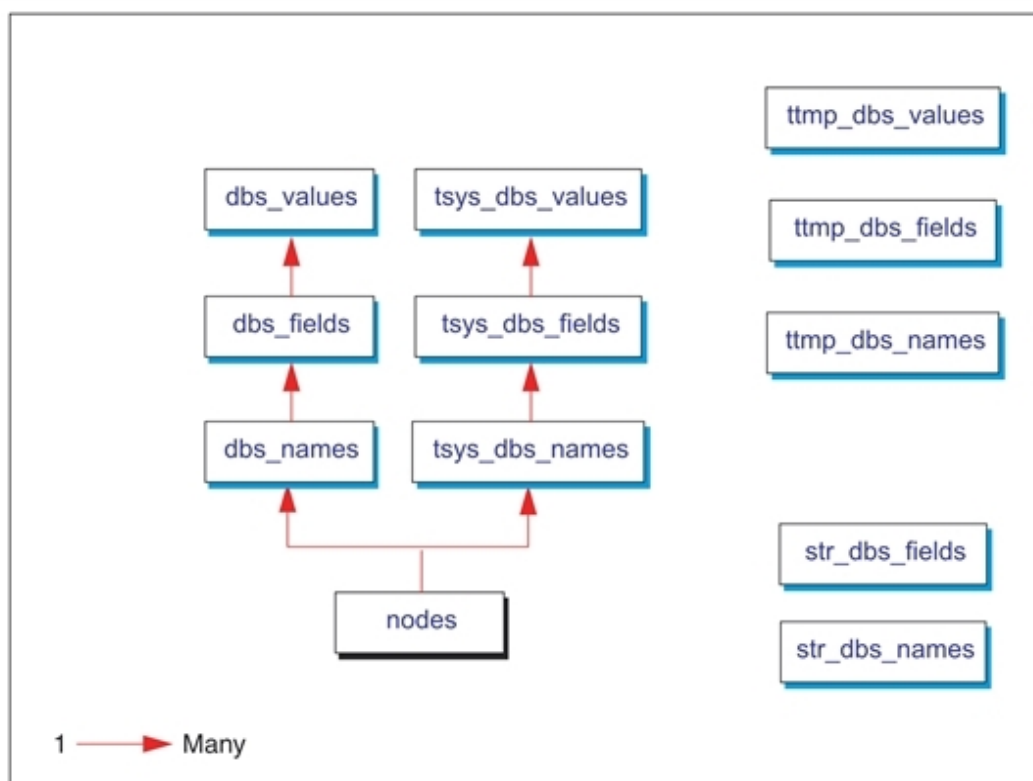
When a Move System Information is performed, if the `tsys_dbs_tables` have been changed, the `dbs_` and/or `str_dbs_` tables are purged and rewritten with the updated information from the `tsys_dbs_tables`.

Access to the `dbs_` and `tsys_dbs_` tables is controlled by a locking mechanism provided by the [flag_table](#) table.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



dbnames

The dbnames table holds the names of all *installed* TIBCO iProcess® Engine tables.

Structure

The dbnames table has the following structure:

```
TABLE dbnames (  
    node_id      INTEGER          NOT NULL,  
    db_id        INTEGER          NOT NULL,  
    db_name      VARCHAR(15)      NOT NULL)
```

Column	Description
node_id	ID of the node that this iProcess Engine table is defined on, as defined in the nodes table.
db_id	Unique ID for this iProcess Engine table.
db_name	Name of this iProcess Engine table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_dbnames	db_id node_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_dbs_names	DELETE	dbs_fields

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_dbs_names_fk	node_id

Table Activity

The `dbs_names` table contains one row for each installed TIBCO iProcess table on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	<p>a Move System Information is performed, if the flag_table indicates that the Tables data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_dbs_names table.</p>
updated	never.
deleted	<p>a Move System Information is performed, if the flag_table indicates that the Tables data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_dbs_names table.</p>

dbf_fields

The `dbf_fields` table holds the field definitions for every field in every *installed* iProcess table.

Structure

The `dbf_fields` table has the following structure:

```
TABLE dbf_fields (
  node_id      INTEGER          NOT NULL,
  dbf_id       INTEGER          NOT NULL,
  field_id     INTEGER          NOT NULL,
  field_name   VARCHAR(15)     NOT NULL,
  field_type   VARCHAR(1)      NOT NULL,
  field_length SMALLINT        NOT NULL,
  field_decimals SMALLINT      NOT NULL)
```

Column	Description
<code>node_id</code>	ID of the node that this field is defined on, as defined in the nodes table.
<code>dbf_id</code>	ID of the table that this field is defined in, as defined in the dbf_names table.
<code>field_id</code>	Unique ID for the field in this TIBCO iProcess® Engine table.
<code>field_name</code>	Name of this field.
<code>field_type</code>	Field type: Either ASCII (A), Numeric (R), Date (D) or Time (T).
<code>field_length</code>	Length of this field, in characters.
<code>field_decimals</code>	Number of characters after the decimal place in this field (relevant only for Numeric fields).

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_dbs_fields	dbs_id field_id node_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_dbs_fields	DELETE	dbs_values

Indexes

The following indexes are defined for this table.

Index Name	Column(s) Indexed
idx_dbs_fields_fk	dbs_id node_id
idx_dbs_fields	field_id dbs_id

Table Activity

The `dbs_fields` table contains one row for each field in each installed TIBCO iProcess table.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a Move System Information is performed, if the flag_table indicates that the Tables data area has been modified.

A row is...	When...
	Note: The table is purged and rewritten using the values from the tsys_dbs_fields table.
updated	never.
deleted	a Move System Information is performed, if the flag_table indicates that the Tables data area has been modified. Note: The table is purged and rewritten using the values from the tsys_dbs_fields table.

dbvalues

The `dbvalues` table holds all field values for all *installed* TIBCO iProcess® Engine tables.

Structure

The `dbvalues` table has the following structure:

```
TABLE dbvalues (
  node_id      INTEGER      NOT NULL,
  dbs_id       INTEGER      NOT NULL,
  record_id    INTEGER      NOT NULL,
  field_id     INTEGER      NOT NULL,
  field_value  VARCHAR(30)  NULL)
```

Column	Description
<code>node_id</code>	ID of the node that this field value is stored on, as defined in the nodes table.
<code>dbs_id</code>	ID of the table that this field value is stored in, as defined in the dbs_names table.
<code>record_id</code>	Unique ID for this record in the iProcess Engine table.

Column	Description
field_id	ID of the field held in this record, as defined in the dbs_fields table.
field_value	Value of the field in this record.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_dbs_values	dbs_id record_id field_id node_id

Triggers

None.

Indexes

The following indexes are defined for this table.

Index Name	Column(s) Indexed
idx_dbs_values_fk	dbs_id field_id node_id
idx_dbs_values	record_id field_id dbs_id

Table Activity

The `db_values` table contains one row for each field of each record in each installed TIBCO iProcess table on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	<p>a Move System Information is performed, if the flag_table indicates that the Tables data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_dbs_values table.</p>
updated	never.
deleted	<p>a Move System Information is performed, if the flag_table indicates that the Tables data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_dbs_values table.</p>

tsys_dbs_names

The `tsys_dbs_names` table is a copy of the [db_names](#) table. It is identical to the [db_names](#) table except for the following:

- The primary key name is `pk_tsys_dbs_names`.
- The following DELETE CASCADE trigger is defined.

Trigger Name	Triggered by	Affects Table(s)
<code>tr_tsys_dbs_names</code>	DELETE	tsys_dbs_fields

- The index name is `idx_tsys_dbs_names_fk`.
- The table is purged and rewritten when a user edits iProcess Engine table data, either by saving changes made in the Table Manager utility in the TIBCO iProcess

Administrator, modifying data with `SWDIR\bin\swutil IMPORT` or `DELTAB`, or by using TIBCO iProcess Objects. (The [flag_table](#) is also updated to indicate that the Tables data area has been modified.)

tsys_dbs_fields

The `tsys_dbs_fields` table is a copy of the [dbs_fields](#) table. It is identical to the [dbs_fields](#) table except for the following:

- The primary key name is `pk_tsys_dbs_fields`.
- The following DELETE CASCADE trigger is defined.

Trigger Name	Triggered by	Affects Table(s)
<code>tr_tsys_dbs_fields</code>	DELETE	tsys_dbs_values

- The index names are `idx_tsys_dbs_fields_fk` and `idx_tsys_dbs_fields`.
- The table is purged and rewritten when a user edits iProcess Engine table data, either by saving changes made in the Table Manager utility in the TIBCO iProcess Administrator, modifying data with `SWDIR\bin\swutil IMPORT` or `DELTAB`, or by using TIBCO iProcess Objects. (The [flag_table](#) is also updated to indicate that the Tables data area has been modified.)

tsys_dbs_values

The `tsys_dbs_values` table is a copy of the [dbs_values](#) table. It is identical to the [dbs_values](#) table except for the following:

- The primary key name is `pk_tsys_dbs_values`.
- The index names are `idx_tsys_dbs_values_fk` and `idx_tsys_dbs_values`.
- The table is purged and rewritten when a user edits iProcess Engine table data, either by saving changes made in the Table Manager utility in the TIBCO iProcess Administrator, modifying data with `SWDIR\bin\swutil IMPORT` or `DELTAB`, or by using TIBCO iProcess Objects. (The [flag_table](#) is also updated to indicate that the Tables data area has been modified.)

str_dbs_names

The `str_dbs_names` table is a copy of the `dbs_names` table. It is identical to the `dbs_names` table except for the following:

- It holds the names of all *uninstalled* TIBCO iProcess® Engine tables.
- The primary key name is `pk_str_dbs_names`.
- No indexes are defined.
- It contains one row for each uninstalled TIBCO iProcess table on the system.

str_dbs_fields

The `str_dbs_fields` table is a copy of the `dbs_fields` table. It is identical to the `dbs_fields` table except for the following:

- It holds the field definitions for every field in every *uninstalled* TIBCO iProcess table.
- The primary key name is `pk_str_dbs_fields`.
- The index name is `idx_str_dbs_fields`.
- It contains one row for each field in each uninstalled TIBCO iProcess table on the system.

ttmp_dbs_names

The `ttmp_dbs_names` table is a temporary copy of the `dbs_names` table. It is identical to the `dbs_names` table except for the following:

- The primary key name is `pk_ttmp_dbs_names`.
- No indexes are defined.
- In most situations the number of rows in the table should be zero.

ttmp_dbs_fields

The `ttmp_dbs_fields` table is a temporary copy of the [dbs_fields](#) table. It is identical to the [dbs_fields](#) table except for the following:

- The primary key name is `pk_ttmp_dbs_fields`.
- No indexes are defined.
- In most situations the number of rows in the table should be zero.

ttmp_dbs_values

The `ttmp_dbs_values` table is a temporary copy of the [dbs_values](#) table. It is identical to the [dbs_values](#) table except for the following:

- The primary key name is `pk_ttmp_dbs_values`.
- No indexes are defined.
- In most situations the number of rows in the table should be zero.

Lists

This section describes the tables that are used to store information about iProcess lists.

About Lists

Note that there are three sets of related tables, as follows:

Prefix	Description
list_	Hold the main system data on iProcess lists, which iProcess Engine references during normal operation.
tsys_ list_	Hold a copy of the main system data, containing users' edits that have not yet been released for use by the system.
ttmp_ list_	Temporary tables used only when importing iProcess lists (using <code>SWDIR\bin\swutil IMPORT</code>).

The `tsys_list_` tables are purged and rewritten whenever a user edits iProcess lists data (either by saving changes made in the List Manager utility in the TIBCO iProcess Administrator, modifying data with `SWDIR\bin\swutil IMPORT`, or by using TIBCO iProcess Objects).

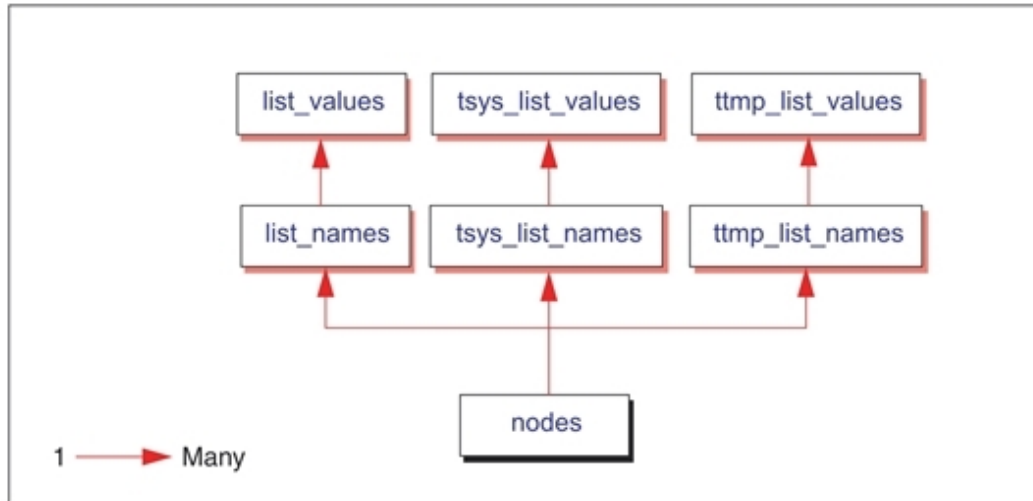
The `list_` tables are purged and rewritten with the updated information from the `tsys_list_` tables when a Move System Information is performed - *if* the [flag_table](#) indicates that the Lists data area has been modified

Access to the `list_` and `tsys_list_` tables is controlled by a locking mechanism provided by the [flag_table](#) table.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



list_names

The `list_names` table holds the names and definitions of all iProcess lists defined on the system.

Structure

The `list_names` table has the following structure:

```

TABLE list_names (
  node_id      INTEGER      NOT NULL,
  list_id      INTEGER      NOT NULL,
  list_name    VARCHAR(15)  NOT NULL,
  list_type    VARCHAR(1)   NOT NULL,
  list_length  INTEGER      NOT NULL,
  list_decimals INTEGER      NOT NULL)
  
```

Column	Description
node_id	ID of the node that this list is stored on, as defined in the nodes table.
list_id	Unique ID for this list.
list_name	Name of this list.
list_type	List type: Either ASCII (A), Numeric (R), Date (D) or Time (T).
list_length	Length of this list, in characters.
list_decimals	Number of characters after the decimal place in this list (relevant only for Numeric lists).

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_list_names	list_id node_id

Triggers

The following DELETE CASCADE trigger is defined for this table.

Trigger Name	Triggered by	Affects Table(s)
tr_list_names	DELETE	list_values

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_list_names_fk	node_id

Table Activity

The `list_names` table contains one row for each iProcess list on the system.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	<p>a Move System Information is performed, if the flag_table indicates that the Lists data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_list_names table.</p>
updated	never.
deleted	<p>a Move System Information is performed, if the flag_table indicates that the Lists data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_list_names table.</p>

list_values

The `list_values` table holds the value of every item in every iProcess list on the system.

Structure

The `list_values` table has the following structure:

```
TABLE list_values (
    node_id      INTEGER      NOT NULL,
    list_id      INTEGER      NOT NULL,
```

record_id	INTEGER	NOT NULL,
list_value	VARCHAR(30)	NULL)

Column	Description
node_id	ID of the node that this list item is stored on, as defined in the nodes table.
list_id	ID of the list that this list item is stored in, as defined in the list_names table.
record_id	Unique ID for this list item.
list_value	Value of this list item.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_list_values	list_id record_id node_id

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_list_values_fk	list_id node_id

Table Activity

The `list_values` table contains one row for each iProcess list item defined on the system. Rows are added, updated and deleted in the following situations.

A row is...	When...
added	<p>a Move System Information is performed, if the flag_table indicates that the Lists data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_list_values table.</p>
updated	never.
deleted	<p>a Move System Information is performed, if the flag_table indicates that the Lists data area has been modified.</p> <p>Note: The table is purged and rewritten using the values from the tsys_list_values table.</p>

tsys_list_names

The `tsys_list_names` table is a copy of the [list_names](#) table. It is identical to the [list_names](#) table except for the following:

- The primary key name is `pk_tsys_list_names`.
- The following DELETE CASCADE trigger is defined.

Trigger Name	Triggered by	Affects Table(s)
<code>tr_tsys_list_names</code>	DELETE	tsys_list_values

- The index name is `idx_tsys_list_names_fk`.
- The table is purged and rewritten when a user edits iProcess list data, either by saving changes made in the List Manager utility in the TIBCO iProcess Administrator, modifying data with `SWDIR\bin\swutil IMPORT`, or by using TIBCO iProcess Objects.

(The [flag_table](#) is also updated to indicate that the Lists data area has been modified.)

tsys_list_values

The `tsys_list_values` table is a copy of the [list_values](#) table. It is identical to the [list_values](#) table except for the following:

- The primary key name is `pk_tsys_list_values`.
- The index name is `idx_tsys_list_values_fk`.
- The table is purged and rewritten when a user edits iProcess list data, either by saving changes made in the List Manager utility in the TIBCO iProcess Administrator, modifying data with `SWDIR\bin\swutil IMPORT`, or by using TIBCO iProcess Objects. (The [flag_table](#) is also updated to indicate that the Lists data area has been modified.)

ttmp_list_names

The `ttmp_list_names` table is a temporary copy of the [list_names](#) table. It is identical to the [list_names](#) table except for the following:

- The primary key name is `pk_ttmp_list_names`.
- The following DELETE CASCADE trigger is defined.

Trigger Name	Triggered by	Affects Table(s)
<code>tr_ttmp_list_names</code>	DELETE	ttmp_list_values

- The index name is `idx_ttmp_list_names_fk`.
- In most situations the number of rows in the table should be zero.

ttmp_list_values

The ttmp_list_values table is a temporary copy of the [list_values](#) table. It is identical to the [list_values](#) table except for the following:

- The primary key name is pk_ttmp_list_values.
- The index name is idx_ttmp_list_values_fk.
- In most situations the number of rows in the table should be zero.

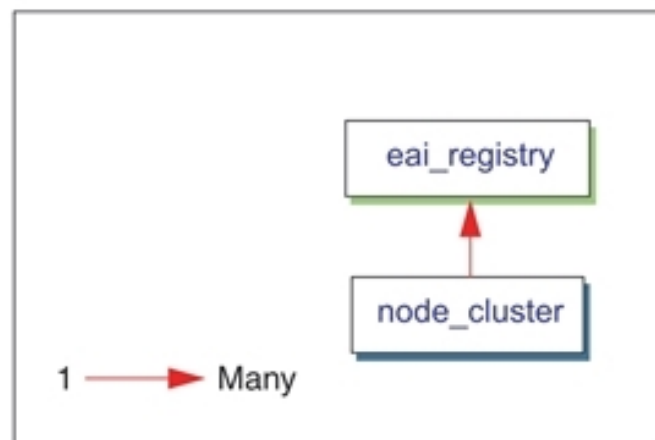
iProcess Server Plug-ins

This section describes the table that is used to store information about iProcess server plug-ins that are installed on this iProcess Engine.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



eai_registry

The `eai_registry` table stores information about each iProcess server plug-in that is installed on this iProcess Engine. The background process reads this table to determine which iProcess server plug-ins to start.

Structure

The eai_registry table has the following structure:

```
TABLE eai_registry(
    eai_type          VARCHAR(20)      NOT NULL,
    logical_machine_id INTEGER        NOT NULL,
    release_version   VARCHAR(32)     NOT NULL,
    plugin_library    VARCHAR(256)    NOT NULL,
    init_params       VARCHAR(1024)   NULL)
```

Column	Description
eai_type	<p>Short name for the EAI Step type that this iProcess server plug-in supports. For example, one of the following:</p> <ul style="list-style-type: none"> EAIDB EAI Database EAISCR EAI Script EAIWEBSERVICES EAI Web Services
logical_machine_id	<p>ID of the computer that this iProcess server plug-in is installed on, as defined in the node_cluster table.</p> <p>Note: If a node cluster architecture is in use, the iProcess server plug-in must be installed on each server in the cluster that is configured to run a background process.</p>
release_version	Version number of this iProcess server plug-in (for example, i10.0-x(3.0)).
plugin_library	Pathname (on this logical_machine_id) where this EAI server plug-in is installed.
init_params	Startup parameters used by this iProcess server plug-in.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_eai_registry	eai_type logical_machine_id

Triggers

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_eai_registry_fk	logical_machine_id

Table Activity

The eai_registry table contains one row for each iProcess server plug-in that is installed on each server in this iProcess Engine node.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	an iProcess server plug-in is installed.
updated	an iProcess server plug-in is upgraded or amended.
deleted	an iProcess server plug-in is deleted.

Firewall Port Ranges

This section describes the tables that store the port range data that is used when the iProcess Engine is used in a firewalled environment.

**Note**

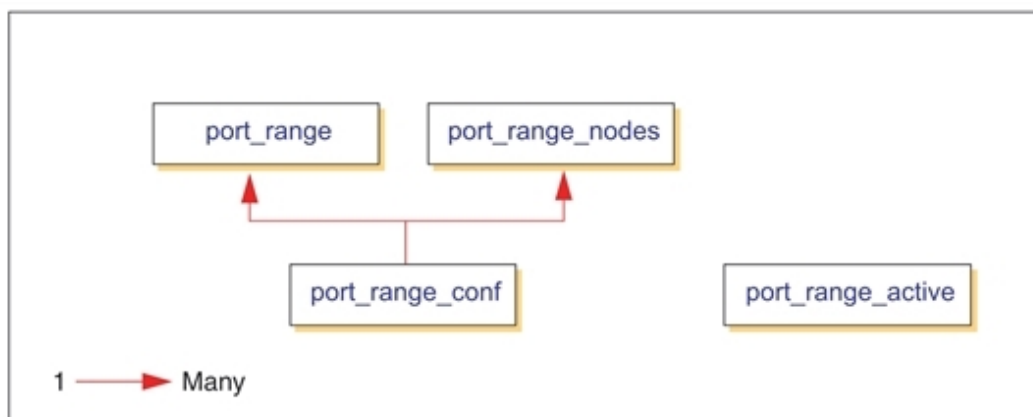
For more information see:

- "Using the iProcess Engine in a Firewalled Environment" in *TIBCO iProcess Engine Architecture Guide*.
- "Administering Firewall Port Ranges" in *TIBCO iProcess Engine Administrator's Guide*.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only trigger-enforced relationships are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



port_range

The `port_range` table contains the firewall data about individual port/RPC numbers that lie within port range configurations defined on this iProcess Engine.

Structure

The `port_range` table has the following structure:

```
TABLE port_range (
    port_range_id          NUMERIC(10)    NOT NULL,
    slot_number            NUMERIC(10)    NOT NULL,
    rpc_number              NUMERIC(10)    NOT NULL,
    port_number             NUMERIC(10)    NOT NULL,
    status                  NUMERIC(10)    NOT NULL,
    logical_machine_id      NUMERIC(10)    NULL,
    logical_process_name    VARCHAR(10)    NULL,
    logical_process_instance NUMERIC(10)    NULL)
```

Column	Description
<code>port_range_id</code>	Unique ID of the port range configuration that this port/RPC number belongs to, as defined in the port_range_conf table.
<code>slot_number</code>	Internal slot in memory used by this port/RPC number.
<code>rpc_number</code>	RPC number.
<code>port_number</code>	Port number.
<code>status</code>	<p>Defines whether this port/RPC number is available or in use by a process. One of the following values:</p> <ul style="list-style-type: none"> • -2 Reserved for future use. • -1 Unobtainable. (A process tried to use the port but found that it was already in use.) • 0 Unallocated. • 1 Allocated to the process defined by the

Column	Description
	logical_machine_id, logical_process_name and logical_process_instance columns.
logical_machine_id	Either: <ul style="list-style-type: none"> ID of the server where the process instance that this port/RPC number has been allocated to runs, as defined in the node_cluster table. 0, if the port/RPC number has not been allocated to a process.
logical_process_name	Logical name of the process instance that this port/RPC number has been allocated to.
logical_process_instance	Unique ID of the process instance that this port/RPC number has been allocated to.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_port_range	port_range_id, slot_number

Triggers

None.

Indexes

None.

Table Activity

The port_range table contains one row per port/RPC number used by the iProcess Engine (if you are using iProcess on a network with a firewall and using port range filtering or

RPC filtering).

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a user defines a new port range configuration, that is, a new record in the port_range_conf table, or modifies the range of an existing port range configuration, using the <i>SWDIR\util\swadm</i> utility.
updated	a process is assigned a slot, or frees up a slot.
deleted	a user deletes a port range configuration, that is, a record in the port_range_conf table, using the <i>SWDIR\util\swadm</i> utility.

port_range_active

The `port_range_active` table lists what port/RPC numbers are being actively used to provide RPC services by iProcess Engine processes.



Note

The table only lists processes that provide RPC services. These processes are `RPC_TCP_LI`, `RPC_UDP_LI`, `RPC_POOL`, `RPC_SWIP`, `WQS` and `WIS`.

Structure

The `port_range_active` table has the following structure:

```
TABLE port_range_active (
    logical_machine_id      INTEGER      NOT NULL,
    logical_process_name    VARCHAR(10)    NOT NULL,
    logical_process_instance INTEGER      NOT NULL,
    process_id              INTEGER      NOT NULL,
    port_number              INTEGER      NOT NULL,
    rpc_number               INTEGER      NOT NULL)
```

Column	Description
logical_machine_id	ID of the server where this process instance runs, as defined in the node_cluster table.
logical_process_name	Logical name of this process instance. Note: See "Administering iProcess Engine Server Processes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of logical process names.
logical_process_instance	Unique ID for this process instance.
process_id	Operating system process ID of this process instance.
port_number	Port number being used by this process instance.
rpc_number	RPC number being used by this process instance.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_port_range_active	logical_machine_id, logical_process_name, logical_process_instance

Triggers

None.

Indexes

None.

Table Activity

The `port_range_active` table contains one row per port/RPC number that is being actively used by the iProcess Engine.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	an iProcess Engine process allocates itself a port/RPC number from either the port_range table or the operating system.
updated	never.
deleted	an iProcess Engine process stops using its assigned port/RPC number, that is, is shut down.

port_range_conf

The `port_range_conf` table defines the available port range configuration(s) for this iProcess Engine, for use with a firewall.



Note

In pre-10.4 iProcess Engine versions this information was defined in the `RNGMODE` parameter of the `SWDIR\etc\staffcfg` file.

Structure

The `port_range_conf` table has the following structure:

```
TABLE port_range_conf (
  port_range_id  INTEGER          NOT NULL,
  range_mode     SMALLINT         NOT NULL,
  range_size     INTEGER          NOT NULL,
  port_start     INTEGER          NOT NULL,
  rpc_start      INTEGER          NOT NULL)
```

Column	Description
port_range_id	Unique ID of this particular port range configuration.
range_mode	<p>Mode used by this port range configuration. One of the following values:</p> <ul style="list-style-type: none"> • 0 No Port or RPC ranging. A process uses the next available port number assigned by the operating system, and an RPC number based on the process ID. • 1 Port ranging. A process uses a port number allocated from within the defined range, and an RPC number based on the process ID. • 2 RPC ranging. A process uses the next available port number assigned by the operating system, and an RPC number allocated from within the defined range. • 3 Port and RPC ranging. A process uses both a port number and an RPC number allocated from within the defined ranges.
range_size	The number of port and RPC numbers allowed in the port number and RPC number ranges.
port_start	The first number in the defined range of port numbers. (The last number = port_start + range_size.)
rpc_start	The first number in the defined range of RPC numbers. (The last number = rpc_start + range_size.)

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_port_range_conf	port_range_id

Triggers

None.

Indexes

None.

Table Activity

The `port_range_conf` table contains one row per defined port range configuration.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a user defines a new port range configuration using the <code>SWDIR\util\swadm</code> utility.
updated	a user changes an existing port range configuration, that is, either mode, range size or starting port/RPC number is changed, using the <code>SWDIR\util\swadm</code> utility.
deleted	a user deletes an existing port range configuration using the <code>SWDIR\util\swadm</code> utility.

port_range_nodes

The `port_range_nodes` table lists which port range configurations (as defined in the [port_range_conf](#) table) are being used by which machines in the iProcess Engine node (as defined in the [node_cluster](#) table).



Note

It is not mandatory for each machine in an iProcess Engine node to have to sit behind the same firewall. Different machines may use different firewalls, or no firewall.

Structure

The `port_range_nodes` table has the following structure:

```
TABLE port_range_nodes (
    port_range_id      INTEGER      NOT NULL,
    logical_machine_id INTEGER      NOT NULL)
```

Column	Description
port_range_id	ID of a particular port range configuration, as defined in the port_range_conf table.
logical_machine_id	ID of the server using this port range configuration, as defined in the node_cluster table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_port_range_nodes	port_range_id, logical_machine_id

Triggers

None.

Indexes

None.

Table Activity

The port_range_nodes table contains one row per server that sits behind a firewall (port range configuration) defined in the [port_range](#) table.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a user adds a machine to the list of servers that sit behind a particular port range configuration, using the <i>SWDIR\util\swadm</i> utility.
updated	a user moves a machine from sitting behind one particular port range configuration to another, using the <i>SWDIR\util\swadm</i> utility.
deleted	a user removes a machine from the list of servers that sit behind a particular port range configuration, using the <i>SWDIR\util\swadm</i> utility.

WQS/WIS Shared Memory

This section describes the [wqs_index](#) table.

Table Relationships

The [wqs_index](#) table has no trigger-enforced relationships with other tables.

wqs_index

The `wqs_index` table holds the information about each work queue on the system that is stored in shared memory by the WQS/WIS processes.

Structure

The `wqs_index` table has the following structure:

```
TABLE wqs_index(
    logical_machine_id      INTEGER      NOT NULL,
    logical_process_instance INTEGER      NOT NULL,
    queue_name              VARCHAR(24)   NOT NULL,
    total_items             NUMERIC(20)   NULL,
    last_cache_time         NUMERIC(20)   NULL,
    new_items               NUMERIC(20)   NULL,
    deadline_items          NUMERIC(20)   NULL,
    urgent_items            NUMERIC(20)   NULL,
    redir_queue_name        VARCHAR(24)   NULL,
    is_cached               SMALLINT      NOT NULL,
    is_group                SMALLINT      NOT NULL,
    is_test                 SMALLINT      NOT NULL,
    is_redirected           SMALLINT      NOT NULL,
    is_disabled             SMALLINT      NOT NULL)
```

Column Name	Description
logical_machine_id	ID of the server where the WIS process that is handling this work queue is running, as defined in the node_cluster table.
logical_process_instance	ID of the instance of the WIS process that is handling this work queue, as defined in the process_config table.
queue_name	Name of the work queue.
total_items	<p>Total number of items in this work queue.</p> <p>Note: When the iProcess Engine starts up the WIS processes use this value to determine whether or not each work queue should be cached. See "Configuring When WIS Processes Cache Their Queues" in <i>TIBCO iProcess Engine Administrator's Guide</i> for more information.</p>
last_cache_time	<p>Either:</p> <ul style="list-style-type: none"> Time taken to cache the work queue (in milliseconds) when it was last cached, either when the WIS process first started up or when the work queue was first accessed. -1, if the work queue has not yet been cached.
new_items	Number of new, unopened items in this work queue.
deadline_items	Number of items in this work queue that have deadlines.
urgent_items	Number of items in this work queue that have an urgent priority.
redir_queue_name	<p>Either:</p> <ul style="list-style-type: none"> the name of the work queue that this queue is currently being redirected to, if the queue is currently being redirected (<code>is_redirected = 1</code>). empty, if the queue is currently not being redirected (<code>is_redirected = 0</code>).

Column Name	Description
is_cached	Indicates whether the queue is currently cached by the WIS process. Either: <ul style="list-style-type: none"> • 1, if the queue is cached. • 0, if the queue is not cached.
is_group	Indicates whether the queue is a Group queue. Either: <ul style="list-style-type: none"> • 1, if the queue is a Group queue. • 0, if the queue is a User queue.
is_test	Indicates whether the queue is a Test queue. Either: <ul style="list-style-type: none"> • 1, if the queue is a Test queue. • 0, otherwise.
is_redirected	Indicates whether the queue is currently being redirected to redir_queue_name. Either: <ul style="list-style-type: none"> • 1, if the queue is currently redirected. • 0, otherwise.
is_disabled	Indicates whether the queue is disabled. Either: <ul style="list-style-type: none"> • 1, if the queue is currently disabled. • 0, otherwise.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_wqs_index	queue_name is_test

Triggers

None

Indexes

None

Table Activity

The `wqs_index` table contains one row for each work queue on the system that is handled by a WIS process.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	a new work queue is allocated to a WIS process by the WQS process.
updated	<ul style="list-style-type: none"> an existing work queue is re-allocated to a different WIS process by the WQS process. a MOVESYSINFO has been processed by the WQS process. the update thread in the WQS process writes the contents of the WQS/WIS shared memory to the database. This update occurs every <code>WQS_PERSIST_SHMEM</code> seconds. <p>Note: See "Administering Process Attributes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for more information about the <code>WQS_PERSIST_SHMEM</code> process attribute.</p>
deleted	a WIS is started as the first time the WIS persists the current shared memory to the database it clears out all existing rows and then writes the shared memory to the database table.

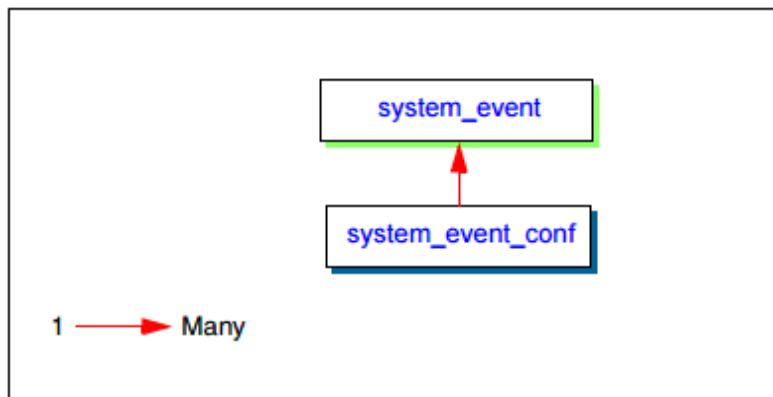
System Event Logging

This section describes the tables that are used to store information about system event logging.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only database-enforced relationships, that is, foreign keys are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



system_event

The `system_event` table contains information about system events.

Structure

The `system_event` table has the following structure:

TABLE system_event			
type_id	number(5)	NOT NULL,	
user_name	varchar(255)	NOT NULL,	
audit_date	date	NOT NULL,	
audit_usecs	number(6)	NOT NULL,	
details	varchar(512)	NOT NULL,	

Column	Description
type_id	IID of an audit event. It is defined in the SWDIR\etc\language.lng\sysevents.cfg file.
user_name	Name of the user who performed the audit event, as defined in the user_names table.
audit_date	Date and time when the event occurred. Note: The audit_usecs column can be combined with this column to provide resolution to a microsecond.
audit_usecs	Number of microseconds since the start of the seconds value specified in the audit_date column.
details	Extra details of the event.

Primary Key

None.

Foreign Keys

None.

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_sys_event_fk	type_id user_name audit_date

Table Activity

The `system_event` table contains one row for each system event that is audited.

Rows are added and deleted in the following situations:

A row is...	When...
added	a new system event is audited.
deleted	using the <code>swadm delete_system_event</code> command to delete system event information or clean system event information that is earlier than a specified period. See <i>TIBCO iProcess Engine Administrator's Guide</i> for more information about the <code>swadm delete_system_event</code> command.

system_event_conf

The `system_event_conf` table holds the configuration information of system events. The information specifies which system event will be audited and published.

Structure

The `system_event_conf` table has the following structure:

TABLE system_event_conf			
type_id	number(5)	NOT NULL,	
audited	number(5)	NOT NULL,	
published	number(5)	NOT NULL,	
event_desc	varchar(256)	NOT NULL,	

Column	Description
type_id	IID of an audit event. It is defined in the SWDIR\etc\language.lng\sysevents.cfg file.
audited	Whether to audit the system event.
published	Whether to publish the system event.
event_desc	Description of the system event.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_event_conf	type_id

Foreign Keys

None.

Indexes

None.

Table Activity

The `system_event_conf` table contains one row for the configuration of each system event.

Rows are added and deleted in the following situations:

A row is...	When...
added	a new system event configuration is imported to iProcess Engine by using the IMPEVENTCONF command. See <i>TIBCO iProcess swutil and swbatch Reference Guide</i> for more information about the IMPEVENTCONF command.
updated	a system event is updated in the configuration file imported by the IMPEVENTCONF command.
deleted	a system event is deleted from the configuration file imported by the IMPEVENTCONF command.

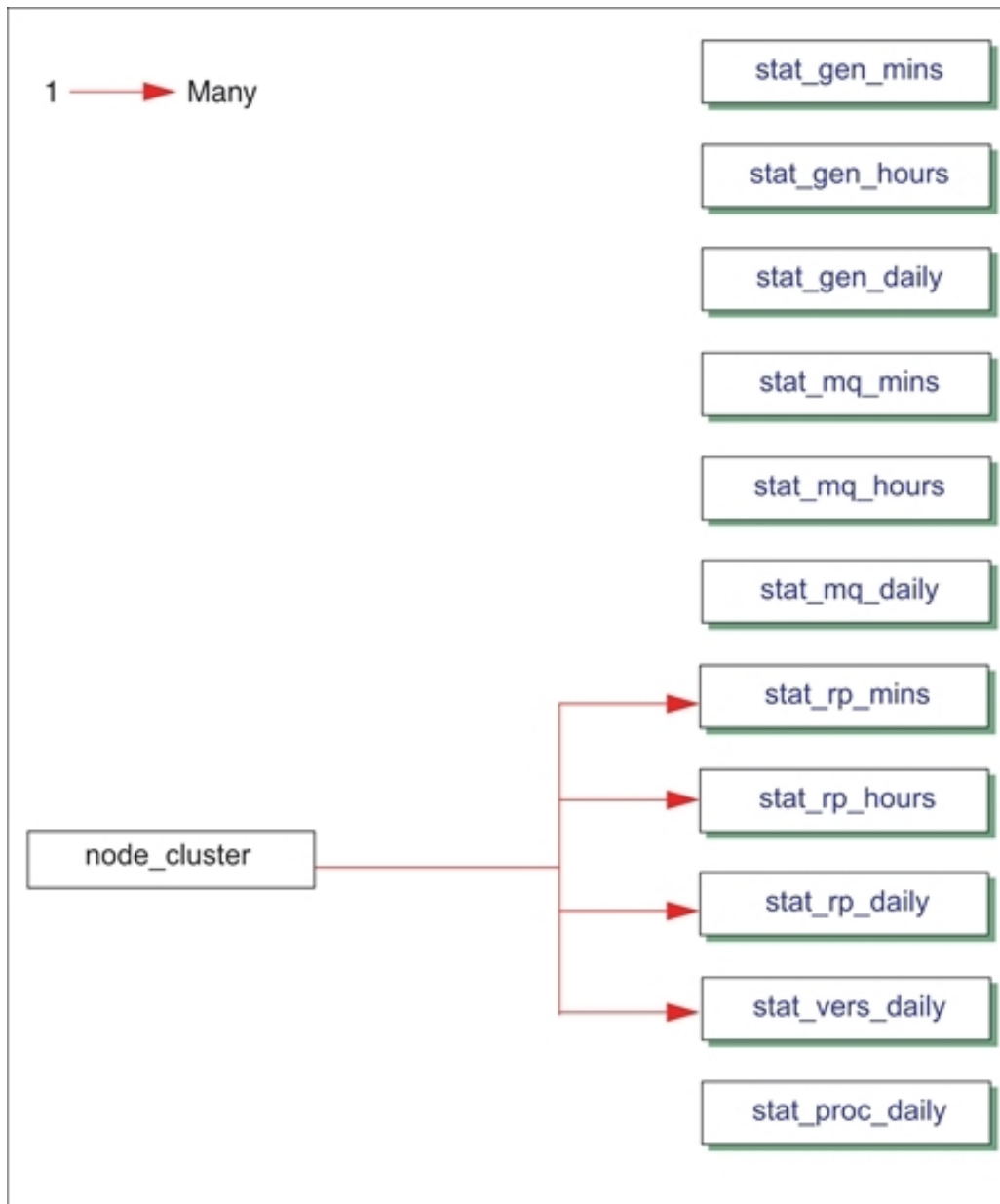
Monitoring

This section describes the database METRICS tables that are used to store information required to monitor system or iProcess Engine performance.

Table Relationships

The following diagram shows how the tables described in this section are related to each other and to other tables in the schema. Note that:

- Only database-enforced relationships, that is, foreign keys, are shown.
- Logical relationships, that is, those used by iProcess, are not shown.



stat_gen_mins

The stat_gen_mins table contains general metrics information on work item server, queues, logins and machine at the minute resolution level.

Structure

The stat_gen_mins table has the following structure:

TABLE stat_gen_mins(
seq_id	INTEGER	,	
cap_date	DATETIME	,	
active_login_count	INTEGER	,	
wis_most_que	INTEGER	,	
wis_most_que_items	INTEGER	,	
most_que_sum	INTEGER	,	
wis_most_item	INTEGER	,	
wis_item_sum	INTEGER	,	
wis_lg_queue	INTEGER	,	
lg_queueuname	VARCHAR_TYPE(32)	,	
lg_queuesize	INTEGER)	

Column	Description
seq_id	Generated sequence number to maintain uniqueness. This is also the Primary Key.
cap_date	Date and time when these details were captured/recorded.
active_login_count	Number of logins active for that minute.

Column	Description
wis_most_que	WIS that is processing most number of Queues.
wis_most_que_items	Work Items count on WIS that is processing most number of Queues.
most_que_sum	Work Items count on WIS that is processing most number of Work Items.
wis_most_item	WIS that is processing most number of Work Items.
wis_item_sum	Work Items count on WIS that is processing most number of Work Items.
wis_lg_queue	WIS that is processing largest Queue
lg_queue_name	Name of largest Queue
lg_queue_size	Size of largest Queue

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_stat_mins_seqid	seq_id

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_gen_m	cap_date

Table Activity

The stat_gen_mins table contains one record for every minute the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every minute.
updated	Never
deleted	After the configured retention period defined by process attribute (value is in days). METRICS_DATA_MINS_KEEP

stat_gen_hours

The stat_gen_hours table contains general metrics information on work item server, cases, users and database at the hour resolution level.

Structure

The stat_gen_hours table has the following structure:

TABLE stat_gen_hours(
seq_id	INTEGER	,
cap_date	DATETIME	,
max_log_count	INTEGER	,
avg_log_count	INTEGER	,

user_count	INTEGER	,
group_count	INTEGER	,
queue_count	INTEGER	,
wis_most_que	INTEGER	,
wis_most_que_items	INTEGER	,
most_que_sum	INTEGER	,
wis_most_item	INTEGER	,
wis_item_sum	INTEGER	,
wis_mi_que_sum	INTEGER	,
wis_lg_queue	INTEGER	,
lg_queueename	VARCHAR_TYPE(32)	,
lg_queuesize	INTEGER	,
outstanding_item_count	INTEGER	,
Dbtiming	INTEGER	,
event_test	VARCHAR_TYPE(128))

Column	Description
seq_id	Generated sequence number to maintain uniqueness. This is also the Primary Key.
cap_date	Date and time when these details were captured/recorded.
max_log_count	Maximum active logins for that hour.

Column	Description
avg_log_count	Average number of logins active for that hour.
user_count	Number of users in iProcess.
group_count	Number of groups in iProcess.
queue_count	Number of queues configured in iProcess.
wis_most_que	WIS that is processing most number of Queues.
wis_most_que_items	Work Items count on WIS that is processing most number of Queues.
most_que_sum	Work Items count on WIS that is processing most number of Work Items.
wis_most_item	WIS that is processing most number of Work Items.
wis_item_sum	Work Items count on WIS that is processing most number of Work Items.
wis_mi_que_sum	Queue count on WIS that is processing most number of Work Items.
wis_lg_queue	WIS that is processing largest Queue
lg_queue_name	Name of largest Queue
lg_queue_size	Size of largest Queue
outstanding_item_count	Number of Work Items pending processing.
Dbtiming	Round trip time to reach or connect to a database.
event_test	Status of event loopback test.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_stat_hour_seqid	seq_id

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_gen_h	cap_date

Table Activity

The stat_gen_hours table contains one record for every hour the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every hour.
updated	Never
deleted	After the configured retention period defined by process attributes (value is in days). METRICS_DATA_HOURS_KEEP

stat_gen_daily

The stat_gen_daily table contains general metrics information on procedures, cases, users, queues and machine at the daily resolution level.

Structure

The stat_gen_daily table has the following structure:

TABLE stat_gen_daily(
seq_id	INTEGER	,	
cap_date	DATETIME	,	
proc_sum	INTEGER	,	
proc_ver_sum	INTEGER	,	
proc_id	INTEGER	,	
proc_name	VARCHAR_TYPE(32)	,	
proc_most_ver	INTEGER	,	
max_casenum	NUMERIC(20)	,	
max_reqid	bigint	,	
case_information_size	NUMERIC(20)	,	
case_data_size	NUMERIC(20)	,	
audit_trail_size	NUMERIC(20)	,	
user_sum	INTEGER	,	
group_sum	INTEGER	,	
queues_sum	INTEGER	,	
max_log_count	INTEGER	,	
avg_log_count	INTEGER	,	
max_outstanding_count	INTEGER	,	

min_outstanding_count	INTEGER	,
avg_outstanding_count	INTEGER	,
max_dbtming	INTEGER	,
min_dbtming	INTEGER	,
avg_dbtming	INTEGER	,
ci_stats_update	DATETIME	,
at_stats_update	DATETIME	,
cd_stats_update	DATETIME)

Column	Description
seq_id	Generated sequence number to maintain uniqueness. This is also the Primary Key.
cap_date	Date and time when these details were captured/recorded.
proc_sum	Total number of procedures.
proc_ver_sum	Total number of versions of all procedures.
proc_id	Id of the procedure with most versions.
proc_name	Name of the procedure with most versions.
proc_most_ver	Number of versions on that procedure with most versions.
max_casenum	Max CASENUM in the system.
max_reqid	Maximum REQID in the system.
case_information_size	Number of rows in Case Information database table

Column	Description
case_data_size	Number of rows in Case Data database table.
audit_trail_size	Number of rows in Audit Trail database table.
user_sum	Number of Users in the system at the start of the day.
group_sum	Number of Groups in the system at the start of the day.
queues_sum	Number of Queues in the system at the start of the day.
max_log_count	Maximum active logins in last 24 hours.
avg_log_count	Average active logins in last 24 hours.
max_outstanding_count	Maximum of <i>Outstanding_Addr</i> count in last 24 hours.
min_outstanding_count	Minimum of <i>Outstanding_Addr</i> count in last 24 hours.
avg_outstanding_count	Average of <i>Outstanding_Addr</i> count in last 24 hours.
max_dbtming	Maximum database connection time in last 24 hours.
min_dbtming	Minimum database connection time in last 24 hours.
avg_dbtming	Average database connection time in last 24 hours.
ci_stats_update	Last status update on Case Information table.
at_stats_update	Last status update on Audit Trail table.
cd_stats_update	Last status update on Case Data table.

Primary Key

The following primary key is defined for this table.

Key Name	Column(s)
pk_stat_day_seqid	seq_id

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_gen_d	cap_date

Table Activity

The stat_gen_daily table contains one record for every minute the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every day.
updated	Never
deleted	After the configured retention period defined by process attributes (value is in days). METRICS_DATA_DAILY_KEEP

stat_mq_mins

The stat_mq_mins table contains general metrics information on total messages in each message queue at the minute resolution level.

Structure

The stat_mq_mins table has the following structure:

TABLE stat_mq_mins(
seq_id	INTEGER	,	
cap_date	DATETIME	,	
queue_id	INTEGER	,	
queue_name	VARCHAR_TYPE(32)	,	
msg_count	INTEGER)	

Column	Description
seq_id	Generated sequence number.
cap_date	Date and time when these details were captured/ recorded.
queue_id	Id of the Message Queue.
queue_name	Name of the Message Queue.
msg_count	Number of Messages in the Queue.

Primary Key

None

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_mq_m	cap_date

Table Activity

The stat_mq_mins table contains one record for every minute the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every minute, one for every Message Queue.
updated	Never
deleted	After the configured retention period defined by process attribute (value is in days). METRICS_DATA_MINS_KEEP

stat_mq_hours

The stat_mq_hours table contains general metrics information on minimum/maximum and average messages in each message queue along with the number of expired messages at the hour resolution level.

Structure

The stat_mq_hours table has the following structure:

TABLE stat_mq_hours(
seq_id	INTEGER	,

cap_date	DATETIME	,
queue_id	INTEGER	,
queue_name	VARCHAR_TYPE(32)	,
max_count	INTEGER	,
min_count	INTEGER	,
avg_count	INTEGER	,
exp_msg_count	INTEGER)

Column	Description
seq_id	Generated sequence number.
cap_date	Date and time when these details were captured/ recorded.
queue_id	ID of the Message Queue.
queue_name	Name of the Message Queue.
max_count	Maximum of the messages count in last hour.
min_count	Minimum of the messages count in last hour.
avg_count	Average of the messages count in last hour.
exp_msg_count	Number of Dead messages in this Message Queue.

Primary Key

None

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_mq_h	cap_date

Table Activity

The stat_mq_hours table contains one record for every hour the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every hour, one per each Message Queue.
updated	Never
deleted	After the configured retention period defined by process attribute (value is in hours). METRICS_DATA_HOURS_KEEP

stat_mq_daily

The stat_mq_daily table contains general metrics information on min/max and average messages in each message queue and the number of expired messages at the day resolution level.

Structure

The stat_mq_daily table has the following structure:

TABLE stat_mq_daily(
seq_id	INTEGER	,

cap_date	DATETIME	,
queue_id	INTEGER	,
queue_name	VARCHAR_TYPE(32)	,
max_count	INTEGER	,
min_count	INTEGER	,
avg_count	INTEGER	,
exp_msg_count	INTEGER)

Column	Description
seq_id	Generated sequence number.
cap_date	Date and time when these details were captured/recorded.
queue_id	Id of the Message Queue
queue_name	Name of the Message Queue
max_count	Maximum of the messages count in 24 hours.
min_count	Minimum of the messages count in 24 hours.
avg_count	Average of the messages count in 24 hours.
exp_msg_count	Number of Dead messages in this Message Queue.

Primary Key

None

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_mq_d	cap_date

Table Activity

The stat_mq_daily table contains one record for every minute the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every day, one for each Message Queue.
updated	Never
deleted	After the configured retention period defined by process attribute (value is in days). METRICS_DATA_DAILY_KEEP

stat_rp_mins

The stat_rp_mins table contains general metrics information on running processes on Process ID, CPU and memory usage, total threads and process start command at the minute resolution level.

Structure

The stat_rp_mins table has the following structure:

TABLE stat_rp_mins(
seq_id	INTEGER	,

cap_date	DATETIME	,
machine_id	INTEGER	,
instance_id	INTEGER	,
process_name	VARCHAR_TYPE(32)	,
pid	INTEGER	,
cputime	VARCHAR_TYPE(32)	,
cpu	INTEGER	,
memory	INTEGER	,
threads	INTEGER	,
starttime	DATETIME	,
command	VARCHAR_TYPE(512))

Column	Description
seq_id	Generated sequence number
cap_date	Date and time when these details were captured/ recorded
machine_id	Machine ID
instance_id	Instance Number of the Process
process_name	Name of the Process
pid	System generated Process ID
cputime	Calculated average CPU usage between two (data collection) intervals.

Column	Description
cpu	CPU Percentage
memory	Memory Usage
threads	Number of threads
starttime	Start time of the process
command	Process start command

Primary Key

None

The following foreign key is defined for this table.

Key Name	Column(s)	Referenced in Table...
fk_stat_pmin_mid	machine_id	node_cluster

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_rp_m	cap_date

Table Activity

The stat_rp_mins table contains one record for every minute the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every minute, one per each Running Process.
updated	Never
deleted	After the configured retention period defined by process attribute (value is in days). METRICS_DATA_MINUTES_KEEP

stat_rp_hours

The stat_rp_hours table contains general metrics information on running processes on min/max and average for CPU and memory usage, total threads and process start command at the hour resolution level.

Structure

The stat_rp_hours table has the following structure:

TABLE stat_rp_hours(
seq_id	INTEGER	,
cap_date	DATETIME	,
machine_id	INTEGER	,
instance_id	INTEGER	,
process_name	VARCHAR_TYPE(32)	,
max_cpu	INTEGER	,
max_mem	INTEGER	,

	max_threads	INTEGER	,
	avg_cpu	INTEGER	,
	avg_mem	INTEGER	,
	avg_threads	INTEGER	,
	min_cpu	INTEGER	,
	min_mem	INTEGER	,
	min_threads	INTEGER)

Column	Description
seq_id	Generated sequence number
cap_date	Date and time when these details were captured/recorded
machine_id	Machine ID
instance_id	Instance Number of the Process
process_name	Name of the Process
max_cpu	Maximum CPU usage in the last hour
max_mem	Maximum memory usage in the last hour
max_threads	Maximum threads count in the last hour
avg_cpu	Average CPU usage in the last hour
avg_mem	Average memory usage in the last hour
avg_threads	Average threads count in the last hour

Column	Description
min_cpu	Minimum CPU usage in the last hour
min_mem	Minimum memory usage in the last hour
min_threads	Minimum threads count in the last hour

Primary Key

The following foreign key is defined for this table.

Key Name	Column(s)	Referenced in Table...
fk_stat_phour_mid	machine_id	node_cluster

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_rp_h	cap_date

Table Activity

The stat_rp_hours table contains one record for every hour the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every hour, one per each Running Process.
updated	Never

A row is...	When...
deleted	After the configured retention period defined by process attribute (value is in days).
	METRICS_DATA_HOURS_KEEP

stat_rp_daily

The stat_rp_daily table contains general metrics information on running processes on min/max and average for CPU and memory usage, total threads and process start command at the daily resolution level.

Structure

The stat_rp_daily table has the following structure:

TABLE stat_rp_daily(
seq_id	INTEGER	,
cap_date	DATETIME	,
machine_id	INTEGER	,
instance_id	INTEGER	,
process_name	VARCHAR_TYPE(32)	,
max_cpu	INTEGER	,
max_mem	INTEGER	,
max_threads	INTEGER	,
avg_cpu	INTEGER	,

avg_mem	INTEGER	,
avg_threads	INTEGER	,
min_cpu	INTEGER	,
min_mem	INTEGER	,
min_threads	INTEGER)

Column	Description
seq_id	Generated sequence number
cap_date	Date and time when these details were captured/recorded
machine_id	Machine ID
instance_id	Instance Number of the Process
process_name	Name of the Process
max_cpu	Maximum CPU usage in the last 24 hours
max_mem	Maximum memory usage in the last 24 hours
max_threads	Maximum threads count in the last 24 hours
avg_cpu	Average CPU usage in the last 24 hours
avg_mem	Average memory usage in the last 24 hours
avg_threads	Average threads count in the last 24 hours
min_cpu	Minimum CPU usage in the last 24 hours
min_mem	Minimum memory usage in the last 24 hours
min_threads	Minimum threads count in the last 24 hours

Primary Key

None

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_rp_d	cap_date

Table Activity

The stat_rp_daily table contains one record for every minute the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every day, one per each Running Process
updated	Never
deleted	After the configured retention period defined by process attribute (value is in days). METRICS_DATA_DAILY_KEEP

stat_vers_daily

The stat_vers_daily table contains daily metrics information on iProcess and Object Server version details.

Structure

The stat_vers_daily table has the following structure:

TABLE stat_vers_daily(
seq_id	INTEGER	,	
cap_date	DATETIME	,	
machine_id	INTEGER	,	
machine_name	VARCHAR_TYPE(256)	,	
iPE_ver	VARCHAR_TYPE(32)	,	
SPO_ver	VARCHAR_TYPE(64)	,	
operating_sys_ver	VARCHAR_TYPE(256))	

Column	Description
seq_id	Generated sequence number
cap_date	Date and time when these details were captured/ recorded.
machine_id	Machine ID
machine_name	Name of the Machine
iPE_ver	iProcess Engine version
SPO_ver	iProcess Objects Server version
operating_sys_ver	OS version details

Primary Key

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_vers_d	cap_date

Table Activity

The stat_vers_daily table contains one record for every minute the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every day
updated	Never
deleted	After the configured retention period defined by process attribute (value is in days). METRICS_DATA_DAILY_KEEP

stat_proc_daily

The stat_proc_daily table contains daily metrics information on procedures and total live and closed cases.

Structure

The stat_proc_daily table has the following structure:

TABLE stat_proc_daily(
seq_id	INTEGER	,

cap_date	DATETIME	,
proc_name	VARCHAR_TYPE(32)	,
flags	VARCHAR_TYPE(32)	,
maj_ver	INTEGER	,
min_ver	INTEGER	,
start_step	VARCHAR_TYPE(32)	,
directory	VARCHAR_TYPE(32)	,
max_casenum	NUMERIC(20)	,
total_cases	INTEGER	,
dead_cases	INTEGER)

Column	Description
seq_id	Generated sequence number.
cap_date	Date and time when these details were captured/recorded.
proc_name	Name of the Procedure
flags	Flags to represent status
maj_ver	Major number in the Procedure version
min_ver	Minor number in the Procedure version
start_step	First step of the Procedure
directory	Procedure directory name

Column	Description
max_casenum	Largest case number of the case created associated to this Procedure
total_cases	Total Number of cases on this Procedure
dead_cases	Number of dead cases on this Procedure

Primary Key

None

Indexes

The following index is defined for this table.

Index Name	Column(s) Indexed
idx_stat_proc_d	cap_date

Table Activity

The stat_proc_daily table contains one record for every hour the server is running and recording data.

Rows are added, updated and deleted in the following situations.

A row is...	When...
added	Rows are added every day
updated	Never
deleted	After the configured retention period defined by process attribute (value is in days). METRICS_DATA_DAILY_KEEP

Views

The following database views are defined for internal use:

- `dbns_nm_fld`
- `tsys_dbns_nm_fld`
- `ttmp_dbns_nm_fld`
- `str_dbns_nm_fld`
- `lst_nm_val`
- `tsys_lst_nm_val`
- `ttmp_lst_nm_val`

**Note**

For more information about these views please see the database creation script (`init2Ksql.sql`).

SSOLite Stored Procedures

This appendix describes the SSOLite stored procedures.

Overview

SSOLite is a set of stored procedures, available in the iProcess database, that provide applications with direct access to a limited subset of iProcess functionality.

An application can use SSOLite stored procedures to issue instructions directly to the iProcess background processes (by inserting messages into the iProcess message queues) to perform the following iProcess operations:

- start a case.
- trigger an event.
- graft a sub-procedure to a procedure (at run-time).
- jump a case to a different point in the procedure.
- suspend a case.
- re-activate a suspended case.

There are four different categories of SSOLite procedure:

- **Data Procedures** are used to create (or clear) any pack data that is required for a particular operation.
- **Command Procedures** are used to perform the iProcess operations described above.
- **Control Procedures** can be used to control the operation of the other SSOLite procedures. Their use is optional.
- **Debug Procedures** can be used to provide debug information about the operation of data and command procedures, if required.

**Warning**

Because of a change in the format of background (BG) process messages, the SSOLite Stored Procedures supplied with this release of the iProcess Engine will not work on versions of the server prior to 10.6.

Using SSOLite Stored Procedures

The following sections discuss some general issues that you need to be aware of when designing an application to use SSOLite stored procedures:

- [Processing Asynchronous Message](#)
- [Transactional Processing](#)
- [Handling Exceptions](#)
- [Processing Queues](#)
- [Prioritizing Messages](#)

Processing Asynchronous Message

SSOLite stored procedures work by sending a message to a database queue, which is processed by one or more background (BG) processes. This means that:

- there is a short delay between an SSOLite stored procedure completing and the BG process processing the instruction.
- even if an SSOLite procedure has completed successfully, the instruction that is processed by the BG may still fail.

Transactional Processing

The BG process will not process any instructions issued by SSOLite stored procedures until the SSOLite transaction has been committed. You can therefore scope transactions according to the requirements of your particular application:

- A transaction can be defined as a single instruction, such as a case start. (If the call to [SW_CASESTART](#) succeeds then a commit is immediately performed.)

- Several instructions can be processed as part of a single transaction. For example, a transaction can add pack data, issue an event, add more pack data and then start several cases, and is only committed if all these operations complete successfully.

Handling Exceptions

SSOLite stored procedures raise a SQL level 16 error message if any procedure fails. Note that:

- The error text is always preceded by the string (SWERROR).
- Each error has a unique ID, which is displayed at the end of the error text.

It is the application's responsibility to handle any such database exceptions, and issue a rollback if appropriate.

The following table describes the different errors (and their unique IDs) that may be returned by the SSOLite stored procedures.



Note

Some of the stored procedures listed in the table are not described in this section. These are lower level stored procedures that may be called by some or all of the stored procedures that are described in this section.

Stored Procedure	SQL Error	Error Text
SW_GET_SEQUENCE_TRANS	50000	Invalid sequence type (<i>seq_type</i>) (ID:001000)
	50000	Unable to create DMO connection, check user is system administrator (ID:001001)
	50000	Unable to set connection type (ID:001002)
	50000	Unable to connect to server errno (<i>Error</i>) (ID:001003) <i>Error</i> is a description of the error returned by the SQL Server sp_OAMethod system stored procedure.
	50000	Unable to verify connection (<i>Return</i>) (ID:001004)

Stored Procedure	SQL Error	Error Text
	50000	Failed to execute sequences - <i>Source</i> (ID:001005) <i>Source</i> is a description of the source of the error.
	50000	Failed to retrieve sequence number - <i>Source</i> (ID:001006) <i>Source</i> is a description of the source of the error.
SW_GET_NODE_DETAILS	50000	Node details not found in database (ID:001007)
	50000	MBox Queue Name(s) not found in database (ID:001008)
SW_GET_SEQUENCE	50000	Failed to find case number (ID:001009)
SW_GET_PROCEDURE	50000	Procedure details not found in database for procedure name= <i>proc_name</i> (ID:001010)
	50000	Procedure version not found in database for procedure name= <i>proc_name</i> , Case Num= <i>case_num</i> (ID:001011)
	50000	Latest Released or Unreleased Procedure version not found in database for procedure name= <i>proc_name</i> (ID:001012)
	50000	Procedure version not found in database for procedure name= <i>proc_name major_version minor_version</i> (ID:001013)
SW_SUSPEND	50000	Suspend type (<i>suspend_type</i>) is invalid, expected 2 (suspend) or 0 (activate) (ID:001014)
	50000	Case (<i>case_num</i>) is already active (ID:001036)
	50000	Case (<i>case_num</i>) is dead (ID:001037)
	50000	Case (<i>case_num</i>) is already suspended (ID:001038)
	50000	Procedure and case information does not match (ID:001042)

Stored Procedure	SQL Error	Error Text
SW_GET_SUBPROC_DETAILS	50000	Sub-Proc casenum not found in database for Procedure <i>proc_name</i> , Case Number <i>case_num</i> , Step Name <i>step_name</i> , Sub-proc <i>sub_proc_name</i> (ID:001018)
SW_DELAYED_RELEASE_ERR	50000	Failed to get node ID (ID:001022)
SW_GETCASE_STATUS_EX	50000	Failed to find case information for case: <i>case_num</i> (ID:001019)
SW_PURGE	50000	Procedure and case information does not match (ID:001041)
SW_CLOSE	50000	Case (<i>case_num</i>) is dead (ID:001039)
	50000	Procedure and case information does not match (ID:001042)

SQL Distributed Management Objects (SQL-DMO)

SSOLite stored procedures access the [sequences](#) table to obtain work item and case number sequence numbers. This locks the table, preventing other iProcess processes from accessing it, for the duration of the transaction. This could cause a problem if, for example, you were batch starting a large number of cases as part of a single transaction.

To prevent this, SSOLite stored procedures use SQL-DMO to connect back to the iProcess database *as a separate transaction* when obtaining sequence numbers.

The use of SQL-DMO means that, when using SSOLite stored procedures, the SQL Server login used to connect to the iProcess database must:

1. use *Windows Authentication* to validate the connection to the iProcess database.
2. have the *Server Administrators SQL Server Role* assigned.

Processing Queues

SSOLite stored procedures write messages to the BG processes using the default background message queues, using a round-robin allocation on a per-session basis. This means that every time a new database session is started the first defined queue (BGMB0X1) is used first. As a result, BGMB0X1 can become overloaded if database sessions are not persisted.

You can override this default behavior for specific transactions by using the [SW_SET_QUEUE](#) and [SW_UNSET_QUEUE](#) control procedures.

Alternatively, you can dedicate specific message queues to handling requests from your SSOLite stored procedure calls. To do this:

1. Create a new Mbox set named SSOLITE. (The Mbox set can use either existing message queues or new ones.)
2. Set the MBSET_WRITE_BG process attribute for your application to assign the SSOLITE1 queue to it. All messages posted to a BG process by the SSOLite stored procedures will now use the SSOLITE Mbox set.

The following example shows a series of commands that you could use to do this.

```
# Add a new SSOLITEQ1 message queue. (Remember to create the
# sw_db_ssolite physical queue first.)
#
swadm add_queue SSOLITEQ1 Local 0003:swpro.sw_db_ssolite

# Add a new SSOLITE Mbox set.
#
swadm add_mboxset SSOLITE Local

# Add the SSOLITEQ1 message queue to the SSOLITE Mbox set (6 is the
# Mboxset ID of the SSOLITE Mboxset).
#
swadm add_queue_to_mboxset 6 SSOLITE1

# Set MBSET_WRITE_BG so that calls from the application's SSOLITE
# stored procedures use the SSOLITE Mbox set to write messages to the
# BG processes.
#
swadm set_attribute 1 SSOLITE 0 MBSET_WRITE_BG 6
#
#Set background processes to read from the queue
#
```

```
swadm add_process 1 BG Y
swadm set_attribute 1 BG 5 MBSET_READ_BG 6
```

**Note**

Because the SSOLite stored procedures cache queue information, you *must* shut down and restart all database connections if you change your message queue configuration in this way.

For more information about message queue configuration, see:

- [Mbox Sets and Message Queues](#).
- "Administering Message Queues and Mbox Sets" in *TIBCO iProcess Engine Administrator's Guide*.

Prioritizing Messages

You can now set priorities ranging from 1 to 999 (where 1 is the highest priority) for internal message queues when passing messages between iProcess processes such as from the background and the WISEs, or from SSOLite to the BG processes. The default message queue priority is 50.

Use the [SW_SET_PRIORITY](#) control procedure to set the internal message queue priorities and the [SW_UNSET_PRIORITY](#) control procedure to restore the default message queue priorities.

The messages with higher internal message queue priorities are processed earlier than those with lower priorities, and the message with the highest priority will automatically be the next message processed, even if there is a backlog in the queue.

If the internal message queue priorities are not set, the messages will be processed in the order of SW_CP_VALUE or SW_IP_VALUE when using iProcess Workspace (Windows) to process work items.

**Note**

When using SSOLite stored procedures to start a case or to trigger an event, the following rules determine which message queue priority settings should be used for processing messages:

- If the value of the SW_CP_VALUE field is set, the message will be processed in

the order of SW_CP_VALUE regardless of the message queue priority that is set by using the SW_SET_PRIORITY control procedure.

- If the SW_CP_VALUE field is not set, the message will be processed in the order of the message queue priority that is set using the SW_SET_PRIORITY control procedure.
- If both the SW_CP_VALUE field and the SW_SET_PRIORITY control procedure are not set for the message priority, the message priority will be set to the default value of the SW_CP_VALUE field, 50.

See *TIBCO iProcess Modeler Advanced Design* for more information about the SW_CP_VALUE field.

Data Procedures

The following data procedures are available:

- [SW_ADD_PACK_DATA](#)
- [SW_ADD_PACK_MEMO](#)
- [SW_CLEAR_PACK_CACHE](#)
- [SW_MODIFY_CASEDATA](#)

SW_ADD_PACK_DATA

The SW_ADD_PACK_DATA procedure defines an item of pack data (a field name/value pair) that will be passed to iProcess with the next command procedure that is called.

Syntax

```
SW_ADD_PACK_DATA (
```

field_name	varchar(31),
------------	--------------

field_value	varchar(255))
-------------	---------------

where:

- `field_name` is a string that specifies the name of the iProcess field that is to be set.
- `field_value` is a string that specifies the value to be set for *field_name*.



Note

Although the value is always passed as a string, it must be in the correct format for the type of field. No validation is performed on either the field name or field value.

Notes

`SW_ADD_PACK_DATA` allows pack data to be passed to iProcess when a command procedure is called:

- You must call `SW_ADD_PACK_DATA` to specify the pack data immediately before calling the desired command procedure.
- A call to `SW_ADD_PACK_DATA` defines a single item of pack data. If you want to define multiple items of pack data, you must make a `SW_ADD_PACK_DATA` call for each piece of data before calling the desired command procedure.
- The pack data is only valid for the next command procedure that is called.

Examples

In the following example, two `SW_ADD_PACK_DATA` calls are used to define data values for the F1 and F2 fields, which are passed to iProcess when Case1 is started (using [SW_CASESTART](#)). The second `SW_CASESTART` call, starting Case2, does not have any data values.

```
EXEC owner.SW_ADD_PACK_DATA 'F1', 'DataItem1'
EXEC owner.SW_ADD_PACK_DATA 'F2', 'DataItem2'
EXEC owner.SW_CASESTART 'CUSTREQ', -1, -1, 'Case1', 'user35', '', 0, 0
EXEC owner.SW_CASESTART 'CUSTREQ', -1, -1, 'Case2', 'user35', '', 0, 0
```

If you want to specify pack data for the F1 and F2 fields for Case2 as well, you must call `SW_ADD_PACK_DATA` again before calling `SW_CASESTART`, as shown .

```
EXEC owner.SW_ADD_PACK_DATA 'F1', 'DataItem1'
EXEC owner.SW_ADD_PACK_DATA 'F2', 'DataItem2'
EXEC owner.SW_CASESTART 'CUSTREQ', -1, -1, 'Case1', 'user35', '', 0, 0
EXEC owner.SW_ADD_PACK_DATA 'F1', 'DataItem1'
```

```
EXEC owner.SW_ADD_PACK_DATA 'F2', 'DataItem2'
EXEC owner.SW_CASESTART 'CUSTREQ', -1, -1, 'Case2', 'user35', '', 0, 0
```

**Note**

These examples do not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_ADD_PACK_MEMO

The SW_ADD_PACK_MEMO procedure defines an item of pack memo data (a field name/value pair) that will be passed to iProcess with the next command procedure that is called.

Syntax

```
SW_ADD_PACK_MEMO (
```

memo_name	varchar(31),
memo_length	integer,
memo_data	varbinary(max),
array_idx	integer =0)

where:

- memo_name is the name of the iProcess memo field (or memo array field).
- memo_length is the number of bytes contained in the memo data.
- memo_data is a raw data field that holds the actual memo data.
- array_idx (optional) can be specified if memo_name is a memo array field; it identifies the specific element in the memo array to be used. If array_idx is not explicitly set, it defaults to a value of 0.

If memo_name is not a memo array field, you should either not set *array_idx*, or set it to 0. (If array_idx contains any other value, no memo data will be found; an error message will be written to the *SWDIR\logs\sw_warn* file.)

Notes

SW_ADD_PACK_MEMO allows pack memo data to be passed to iProcess when a command procedure is called:

- You must call SW_ADD_PACK_MEMO to specify the pack memo data immediately before calling the desired command procedure.
- A call to SW_ADD_PACK_MEMO defines a single item of pack memo data. If you want to define multiple items of pack memo data, you must make a SW_ADD_PACK_MEMO call for each piece of memo data before calling the desired command procedure.
- The pack memo data is only valid for the next command procedure that is called.

Example

In the following example, two SW_ADD_PACK_MEMO calls are used to define memo data values for the two fields, which are passed to iProcess when Case1 is started.

```
begin
declare @mvalue as varbinary(max)
declare @aa as varchar(max)
set @aa = '111111'
set @mvalue = cast(@aa as varbinary(max))
EXEC swpro1.SW_ADD_PACK_MEMO 'MEMO1',6, @mvalue,0
EXEC swpro1.SW_ADD_PACK_MEMO 'MEMO2',6, @mvalue,0
EXEC owner.SW_CASESTART 'CUSTREQ', -1, -1, 'Case1', 'user35', '', 0, 0
end
```



Note

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_CLEAR_PACK_CACHE

The SW_CLEAR_PACK_CACHE procedure clears any items of pack data or pack memo data that have been added using [SW_ADD_PACK_DATA](#) or [SW_ADD_PACK_MEMO](#) calls, prior to calling a command procedure.

Syntax

```
SW_CLEAR_PACK_CACHE ()
```

Notes

Use SW_CLEAR_PACK_CACHE if added data is no longer required.

SW_MODIFY_CASEDATA

The SW_MODIFY_CASEDATA procedure allows you to modify the data of an existing case. Use an [SW_ADD_PACK_DATA](#) procedure to specify the data to be modified. Then, an immediately following SW_MODIFY_CASEDATA posts an instruction to the BG process to carry out the change. You can use the SW_MODIFY_CASEDATA procedure to set case data for main procedures and sub-procedures.

This event is audited, using audit message 058. See *TIBCO iProcess Engine Administrator's Guide* for details of audit messages.

Syntax

```
SW_MODIFY_CASEDATA (
```

proc_name	varchar(8),
proc_maj_ver	integer,
proc_min_ver	integer,
case_number	numeric(20),
reason	varchar(24),
user_id	varchar(24))

where:

- `proc_name` is the name of the procedure that you want to modify a case of.
- `proc_maj_ver` is either the major version number of the `proc_name` procedure, or `-1`. See the notes .
- `proc_min_ver` is either the minor version number of the `proc_name` procedure, or `-1`. See the notes .
- `case_number` is the case number of the main procedure for which the data is to be modified.
- `reason` is a reason for the case data modification, used in the audit trail.
- `user_id` is the name of the iProcess user who is performing the modification.

Notes

Instead of using the specific major and/or minor version number of the procedure, you can specify both the `proc_maj_ver` and `proc_min_ver` parameters as `-1`. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as `-1`, you must specify the other one as `-1` as well.

Example

This example modifies data for case 876 of the Transfer procedure. The `SW_ADD_PACK_DATA` statement changes the value of the `TEXT1` field to "New customer name". The `SW_MODIFY_CASEDATA` call then identifies the procedure and case to be changed, and provides the "Modified For Graft" message which will be displayed in the audit trail.

```
EXEC swpro.SW_ADD_PACK_DATA 'TEXT1', 'New customer name'
EXEC swpro.SW_MODIFY_CASEDATA 'Transfer', -1, -1, 876, 'Modified For
Graft', 'swadmin'
```

Command Procedures

The following command procedures are available:

- [SW_AUDIT](#)
- [SW_CASEREOPEN](#)
- [SW_CASESTART](#)
- [SW_CLOSE](#)
- [SW_CLOSE_WITHOUT_EVENT](#)
- [SW_DELAYED_RELEASE_ERR](#)
- [SW_EVENT](#)
- [SW_EVENT_UPDATE_PACK](#)
- [SW_GETCASE_STATUS_EX](#)
- [SW_GETCASE_STATUS](#)
- [SW_GRAFT](#)
- [SW_GRAFTCOUNT](#)
- [SW_JUMPTO](#)
- [SW_JUMPTO_MULTI](#)
- [SW_PURGE](#)
- [SW_PURGE_WITHOUT_EVENT](#)
- [SW_SUSPEND](#)
- [SW_ACTIVATE](#)

SW_AUDIT

The `SW_AUDIT` procedure instructs the iProcess Engine background (BG) process to create the specified audit trail message for the specified case.

Syntax

SW_AUDIT (

<i>proc_name</i>	<code>varchar(8),</code>
<i>proc_maj_ver</i>	<code>integer,</code>
<i>proc_min_ver</i>	<code>integer,</code>
<i>case_num</i>	<code>numeric(20) output,</code>
<i>Audit_id</i>	<code>integer</code>
<i>Audit_step</i>	<code>varchar(8)</code>
<i>Audit_desc</i>	<code>varchar(24)</code>
<i>User_id</i>	<code>varchar(255))</code>

where:

- *proc_name* is the name of the procedure that you want to create an audit message for.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *case_num* (input) is the name of a variable, defined in the calling program, into which SW_AUDIT will return the case number of the started case. If this information is not required, specify this parameter as 0.
- *Audit_id* is the numeric value of the audit message required. User audit messages will be values greater than 256, as listed in the `SWDIR/etc/english.lng/auditusr.mes` file. See "Understanding Audit Trails" in *TIBCO iProcess Engine Administrator's Guide* for details.
- *Audit_step* is the stepname of this audit. If the step is not required for this audit message, specify this parameter as a null string ("") instead.
- *Audit_desc* is the description to be added to the audit message.

- *User_id* is the username that will be added to the audit trail entry.

Notes

Instead of using the specific major and/or minor version number of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

Example

This example creates an audit message 131 for the CARPOOL procedure.

```
EXEC swpro.SW_AUDIT 'CARPOOL', -1, -1, 99999, 131, '', 'BW Activity', 'BW
User'
```

SW_CASEREOPEN

The SW_CASEREOPEN procedure resurrects a case.

Syntax

```
SW_CASEREOPEN (
  proc_name      varchar(8),
  user_id        varchar(24),
  step_name      varchar(8),
  case_num       numeric(20))
```

where:

- *proc_name* is the name of the procedure that you want to resurrect.
- *user_id* is the name of the iProcess user who is resurrecting the case.

- *step_name* is the name of the case step that you want to resurrect.
- *case_num* is the number of the case that you want to resurrect.

Notes

After a case is closed, all the deadlines of the case are removed. If the case is reopened, you can reset the deadlines by running the `CreateCaseDeadline` function. For more information about the `CreateCaseDeadline` function, see *TIBCO iProcess Expressions and Functions Reference Guide*.

Example

This example resurrects step STEP1 of case 101 of procedure CUSTREQ.

```
EXEC ssolite.SW_CASEREOPEN 'CUSTREQ', 'user35', 'STEP1', 101
```

SW_CASESTART

The `SW_CASESTART` procedure starts a case of a procedure.

Syntax

`SW_CASESTART (`

<i>proc_name</i>	<code>varchar(8),</code>
------------------	--------------------------

<i>proc_maj_ver</i>	<code>integer,</code>
---------------------	-----------------------

<i>proc_min_ver</i>	<code>integer,</code>
---------------------	-----------------------

<i>case_desc</i>	<code>varchar(24),</code>
------------------	---------------------------

<i>user_id</i>	<code>varchar(24),</code>
----------------	---------------------------

<i>step_name</i>	<code>varchar(8),</code>
------------------	--------------------------

<i>case_num</i>	numeric(20) output,
<i>request_id</i>	numeric(20) output)

where:

- *proc_name* is the name of the procedure that you want to start a case of.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *case_desc* is a suitable description for this case.
- *user_id* is the name of the iProcess user who is starting the case.
- *step_name* is the name of the step at which the case should start. If you want to use the default start step, specify this parameter as a null string ('').
- *case_num* (output) is the name of a variable, defined in the calling program, into which SW_CASESTART will return the case number of the started case. If this information is not required, specify this parameter as 0.
- *request_id* (output) is the name of a variable, defined in the calling program, into which SW_CASESTART will return the REQ ID of the work item that is sent out when the case is started. If this information is not required, specify this parameter as 0.

Notes

Instead of using the specific major and/or minor version number of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will determine which version of the procedure to use according to the following rules:

1. the latest released version of the procedure or, if no released version exists,
2. the latest unreleased version of the procedure.



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

Example

This example starts a case of the CUSTREQ procedure. Note that pack data values for the CustName and CustID fields are provided by separate calls to [SW_ADD_PACK_DATA](#) immediately before the SW_CASESTART call.

```
EXEC owner.SW_ADD_PACK_DATA 'CustName', 'Allsop, J.A'
EXEC owner.SW_ADD_PACK_DATA 'CustID', '478163'
EXEC owner.SW_CASESTART 'CUSTREQ', -1, -1, 'Refund request', 'user35', '',
0, 0
```

SW_CLOSE



Note

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

The SW_CLOSE procedure closes an active case of a procedure.

If an event is set for the OnBeforeClose event, the event will be triggered when the case is about to close but before the case is actually closed. If an event is set for the OnAfterClose event, the event will be triggered after closing the case.

Syntax

SW_CLOSE (

<i>proc_name</i>	<i>varchar(8),</i>
<i>proc_maj_ver</i>	<i>integer,</i>
<i>proc_min_ver</i>	<i>integer,</i>
<i>case_number</i>	<i>numeric(20),</i>
<i>user_id</i>	<i>varchar(24))</i>

where:

- *proc_name* is the name of the procedure that you want to close a case of.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *case_num* is the number of the case that is to be closed.
- *user_id* is the name of the iProcess user who is closing the case.

Notes

Instead of using the specific major and/or minor version number of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).

i Note: If you specify one version number parameter as -1, you must specify the other one as -1 as well.



Note

Example

This example closes the 103 case of the CUSTREQ procedure without triggering an event.

```
EXEC owner.SW_CLOSE_WITHOUT_EVENT 'CUSTREQ', -1, -1, 103, 'swadmin'
```



Note

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_CLOSE_WITHOUT_EVENT

The SW_CLOSE_WITHOUT_EVENT procedure closes an active case of a procedure without triggering the events that are set for the OnBeforeClose event or the OnAfterClose event.

Syntax

SW_CLOSE_WITHOUT_EVENT (

<i>proc_name</i>	<code>varchar(8),</code>
<i>proc_maj_ver</i>	<code>integer,</code>
<i>proc_min_ver</i>	<code>integer,</code>
<i>case_number</i>	<code>numeric(20),</code>
<i>user_id</i>	<code>varchar(24))</code>

where:

- *proc_name* is the name of the procedure that you want to close a case of.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *case_num* is the number of the case that is to be closed.
- *user_id* is the name of the iProcess user who is closing the case.

Notes

Instead of using the specific major and/or minor version number of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).

**Note**

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

Example

This example closes the 103 case of the CUSTREQ procedure without triggering an event.

```
EXEC owner.SW_CLOSE_WITHOUT_EVENT 'CUSTREQ', -1, -1, 103, 'swadmin'
```

**Note**

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_DELAYED_RELEASE_ERR

The SW_DELAYED_RELEASE_ERR procedure takes a BG action when a delayed release error occurs.

Syntax

SW_DELAYED_RELEASE_ERR (

<i>delayed_release_id</i>	<code>varchar(256),</code>
<i>audit_desc</i>	<code>varchar(255),</code>
<i>user_id</i>	<code>varchar(24),</code>
<i>err_code</i>	<code>varchar(20),</code>
<i>err_message</i>	<code>varchar(255),</code>
<i>bg_action</i>	<code>integer)</code>

where:

- *delayed_release_id* is the ID of the delayed release.
- *audit_desc* is the description of the delayed release audit.
- *user_id* is the user ID.
- *err_code* is the error code.
- *err_message* is the error message.
- *bg_action* is the BG action to handle the delayed release error.

Notes

If a delayed release error occurs, TIBCO BusinessWorks returns an error code, an error message, and a BG action to TIBCO iProcess Engine. The BG process take the following actions according to BG action:

- Just log the error and do nothing.
- Log the error and requeue the transaction again.
- Log the error and progress the iProcess case, as if it has been released.

See *TIBCO iProcess Connector for ActiveMatrix BusinessWorks User's Guide* for more information about the delayed release error.

SW_EVENT

The SW_EVENT procedure triggers a specific event on a case of a procedure.

Syntax

SW_EVENT (

<i>proc_name</i>	<code>varchar(8),</code>
<i>proc_maj_ver</i>	<code>integer,</code>
<i>proc_min_ver</i>	<code>integer,</code>

<i>step_name</i>	<code>varchar(8),</code>
<i>case_num</i>	<code>numeric(20),</code>
<i>user_id</i>	<code>varchar(24))</code>

where:

- *proc_name* is the name of the procedure that you want to trigger the event on.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *step_name* is the name of the event step that you want to trigger.
- *case_num* is the number of the case that you want to trigger the event on.
- *user_id* is the name of the iProcess user who is triggering the event.

Notes

Instead of using the specific major or minor version number or both of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

For more information about events and how to use them, see *TIBCO iProcess Modeler Integration Techniques*.

Example

This example issues an event, as user `swadmin`, on step `STEP1` of case `101` of the `CUSTREQ` procedure.

```
EXEC owner.SW_EVENT 'CUSTREQ', -1, -1, 'STEP1', 101, 'swadmin'
```

**Note**

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_EVENT_UPDATE_PACK

The SW_EVENT_UPDATE_PACK procedure is the same as [SW_DELAYED_RELEASE_ERR](#), but when it triggers a specific event on a case of a procedure it refreshes the data of any work items that are outstanding for that case.

Syntax

```
SW_EVENT_UPDATE_PACK (
```

<i>proc_name</i>	<code>varchar(8),</code>
<i>proc_maj_ver</i>	<code>integer,</code>
<i>proc_min_ver</i>	<code>integer,</code>
<i>step_name</i>	<code>varchar(8),</code>
<i>case_num</i>	<code>numeric(20),</code>
<i>user_id</i>	<code>varchar(24))</code>

where:

- *proc_name* is the name of the procedure that you want to trigger the event on.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .

- *step_name* is the name of the event step that you want to trigger.
- *case_num* is the number of the case that you want to trigger the event on.
- *user_id* is the name of the iProcess user who is triggering the event.

Notes

Instead of using the specific major or minor version number or both of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

For more information about events and how to use them, see *TIBCO iProcess Modeler Integration Techniques*.

Example

This example issues an event, as user *swadmin*, on step *STEP1* of case 101 of the *CUSTREQ* procedure, and refreshes outstanding work items.

```
EXEC owner.SW_EVENT_UPDATE_PACK 'CUSTREQ', -1, -1, 'STEP1', 101, 'swadmin'
```



Note

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_GETCASE_STATUS

The *SW_GETCASE_STATUS* procedure returns the status of a case of a procedure.

Syntax

SW_GETCASE_STATUS (

<i>case_num</i>	numeric(20),
<i>case_status</i>	varchar(10) output,
<i>proc_type</i>	varchar(10)output,
<i>case_started</i>	datetime output)

where:

- *case_num* is the number of the case that you want to get the status of.
- *case_status* (output) is the name of a variable, defined in the calling program, into which SW_GETCASE_STATUS will return the status of the specified case.
- *proc_type* (output) is the name of a variable, defined in the calling program, into which SW_GETCASE_STATUS will return the procedure type of the specified case (for example, Main or Sub).
- *case_started* (output) is the name of a variable, defined in the calling program, into which SW_GETCASE_STATUS will return the date and time that the case was started.

Example

This example displays the status of case 8.

```
Declare @case_status as varchar(10)
Declare @proc_type as varchar(10)
Declare @case_started as datetime
EXEC owner.SW_GETCASE_STATUS 8, @case_status output, @proc_type output,
@case_started output
print 'Case status :'+@case_status
print 'Proc Type :'+@proc_type
print 'Case started : '+ cast(@case_started as varchar)
```

This results in output displaying the status of the case. For example:

```
Case status :Active  
Proc type :Main  
Case started : MAY 25 2005 3:36PM
```

**Note**

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_GETCASE_STATUS_EX

The SW_GETCASE_STATUS_EX procedure retrieves the case status of a procedure.

Syntax

SW_GETCASE_STATUS_EX (

<i>proc_num</i>	numeric(20),
<i>case_num</i>	numeric(20),
<i>case_status</i>	varchar _TYPE(10))
<i>proc_type</i>	varchar _TYPE(5),
<i>case_started</i>	timestampdatetime,

where:

- *proc_num* is the procedure id for which the status needs to be retrieved.
- *case_number* is the number of the case for which status needs to be retrieved.
- *case_status* is the return value for case status. It can be one of the three values: Active, Suspended, or Dead.
- *proc_type* defines the process type. The value is either Sub, or Main.
- *case_started* returns the date and time when the case started.

**Note**

SW_GETCASE_STATUS_EX is similar to SW_GETCASE_STATUS except that it takes an extra parameter `proc_num` as one of the input parameters.

**Note**

In Process Engine versions prior to 10, case numbers were unique only under a given Procedure, but become duplicate at the Engine level. Since Procedures are always identified by `proc_num`, in such situations, a `proc_num` is mandatory, to uniquely identify a case. SW_GETCASE_STATUS_EX is used in such a situation instead of SW_GETCASE_STATUS as it takes `proc_num` and `case_num` as inputs.

SW_GRAFT

The SW_GRAFT procedure grafts a sub procedure onto a graft step in a main procedure. The case data is added to the sub-procedure.

Syntax

SW_GRAFT (

<i>proc_name</i>	<code>varchar(8),</code>
<i>proc_maj_ver</i>	<code>integer,</code>
<i>proc_min_ver</i>	<code>integer,</code>
<i>case_number</i>	<code>numeric(20)output,</code>
<i>graft_step_name</i>	<code>varchar(8),</code>
<i>graft_proc_name</i>	<code>varchar(8),</code>
<i>graft_proc_maj_ver</i>	<code>integer,</code>
<i>graft_proc_min_ver</i>	<code>integer,</code>
<i>graft_id</i>	<code>varchar(49))</code>

where:

- *proc_name* is the name of the parent procedure that you want to graft a sub-procedure to.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *case_number* is the number of the case that you want to graft a sub-procedure to.
- *graft_step_name* is the name of the graft step in the *proc_name* procedure that the sub-procedure is to be grafted to.
- *graft_proc_name* is the name of the sub-procedure that is to be grafted to the *proc_name* parent procedure.
- *graft_proc_maj_ver* is either the major version number of the *graft_proc_name* procedure, or -1. See the notes
- *graft_proc_min_ver* is either the minor version number of the *graft_proc_name* procedure, or -1. See the notes
- *graft_id* is a unique identifier for this instance of the *graft_step_name* graft step.

Notes

Instead of using the specific major and/or minor version number of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

For more information about graft steps and how to use them, see *TIBCO iProcess Modeler Integration Techniques*.

Example

This example uses SW_GRAFT to graft the SUBPROC1 sub-procedure to step GRAFT01 of case 101 of the CUSTREQ procedure. It then uses SW_GRAFTCOUNT to specify that a single item is to be grafted to the UNIQUEID instance of the graft step.

```
EXEC owner.SW_GRAFT 'CUSTREQ', -1, -1, 101, 'GRAFT01', 'SUBPROC1', -1, -1,
'UNIQUEID'
EXEC owner.SW_GRAFTCOUNT 'CUSTREQ', -1, -1, 101, 'GRAFT01', 'UNIQUEID', 1
```



Note

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_GRAFTCOUNT

The SW_GRAFTCOUNT procedure specifies how many items are to be grafted to the specified instance of the graft step.

Syntax

SW_GRAFTCOUNT (

<i>proc_name</i>	<code>varchar(8),</code>
<i>proc_maj_ver</i>	<code>integer,</code>
<i>proc_min_ver</i>	<code>integer,</code>
<i>case_number</i>	<code>numeric(20)output,</code>
<i>graft_step_name</i>	<code>varchar(8),</code>
<i>graft_id</i>	<code>varchar(49),</code>
<i>graft_count</i>	<code>integer)</code>

where:

- *proc_name* is the name of the parent procedure that you want to graft a sub-procedure to.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *case_number* is the case number of the main procedure that the sub-procedure is to be grafted to.
- *graft_step_name* is the name of the graft step in the *proc_name* procedure that the sub-procedure is to be grafted to.
- *graft_id* is a unique identifier for this instance of the *graft_step_name* graft step.
- *graft_count* is the number of items that are to be grafted to the *graft_step_name* graft step.

Notes

Instead of using the specific major or minor version number or both of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

For more information about graft steps and how to use them, see *TIBCO iProcess Modeler Integration Techniques*.

Example

See [SW_GRAFT](#).

SW_JUMPTO

The SW_JUMPTO procedure jumps a case from its current step to another step in the procedure, ignoring the procedure logic.

Syntax

SW_JUMPTO (

<i>proc_name</i>	<code>varchar(8),</code>
<i>proc_maj_ver</i>	<code>integer,</code>
<i>proc_min_ver</i>	<code>integer,</code>
<i>jump_step</i>	<code>varchar(8),</code>
<i>case_number</i>	<code>numeric(20),</code>
<i>jump_reason</i>	<code>varchar(24),</code>
<i>user_id</i>	<code>varchar(24)</code>
<i>failurecount</i>	<code>in integer,)</code>

where:

- *proc_name* is the name of the procedure that you want to jump a case of.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *jump_step* is the name of the step that the case is to jump to.
- *case_number* is the case number of the main procedure that is to jump.
- *jump_reason* is a reason for this jump, used in the audit trail
- *user_id* is the name of the iProcess user who is performing the jump.

- *failurecount* is an optional parameter. If passed, is used internally to pass it on to BusinessWorks and to wait for some time before retry.

Notes

Instead of using the specific major or minor version number or both of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

If a SW_JUMPTO procedure specifies an invalid *jump_step*, the transaction is rolled back. A warning message is generated and an Invalid Step message is written to the audit trail.

For more information about jumps and how to use them, please see the TIBCO iProcess Objects and TIBCO iProcess Server Objects programmer guides.

Example

This example jumps case 102 of the CUSTREQ procedure from its current position in the workflow to STEP5. The reason for the jump will be displayed in the audit trail as “Administrator-initiated Jump”.

```
EXEC owner.SW_JUMPTO 'CUSTREQ', -1, -1, 'STEP5', 102, 'Administrator-
initiated Jump', 'swadmin'
```



Note

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_JUMPTO_MULTI

The SW_JUMPTO_MULTI procedure is similar to [SW_JUMPTO](#) except that it can process, that is, jump to, more than one step. It allows the withdrawal of either a single step or all

steps. In addition it allows setting of case data using the existing [SW_ADD_PACK_DATA](#) interface.

Syntax

SW_JUMPTO_MULTI (

<i>tgt_proc_name</i>	<code>varchar(8),</code>
<i>tgt_proc_maj_ver</i>	<code>integer,</code>
<i>tgt_proc_min_ver</i>	<code>integer,</code>
<i>src_step_name</i>	<code>varchar(8),</code>
<i>tgt_step_name</i>	<code>varchar(1024),</code>
<i>case_number</i>	<code>numeric(20),</code>
<i>jump_reason</i>	<code>varchar(24),</code>
<i>user_id</i>	<code>varchar(24))</code>

where:

- *tgt_proc_name* is the name of the procedure that you want to jump a case of.
- *tgt_proc_maj_ver* is either the major version number of the *tgt_proc_name* procedure, or -1. See the notes .
- *tgt_proc_min_ver* is either the minor version number of the *tgt_proc_name* procedure, or -1. See the notes .
- *src_step_name* is the name of the step to be withdrawn. Specifying * withdraws all outstanding steps.
- *tgt_step_name* is the name of the step that the case is to jump to. Use a comma-separated list of step names to jump to more than one step.
- *case_number* is the case number of the main procedure that is to jump.
- *jump_reason* is a reason for this jump, used in the audit trail.
- *user_id* is the name of the iProcess user who is performing the jump.

Notes

Instead of using the specific major or minor version number or both of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

If a `SW_JUMPTO_MULTI` procedure specifies an invalid *jump_step*, the transaction is rolled back. A warning message is generated and an Invalid Step message is written to the audit trail.

For more information about jumps and how to use them, please see the TIBCO iProcess Objects and TIBCO iProcess Server Objects programmer guides.

Example

This example jumps case 110 of the `CARPOOL` procedure to the `ALLOCATE` steps and `REFUSED`. The `REQUEST` step is withdrawn. The reason for the jump will be displayed in the audit trail as “Request Refused”.

```
EXEC owner.SW_JUMPTO_MULTI 'CARPOOL', -1, -1, 'REQUEST',
'ALLOCATE,REFUSED', 110, 'Request Refused', 'swadmin'
```



Note

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_PURGE

The `SW_PURGE` procedure purges the specified case of a procedure (permanently deleting it from the system). If events are set for the `OnBeforePurge` event, the events will be triggered when the case is about to purge but before the case is actually purged.

Syntax

SW_PURGE (

<i>proc_name</i>	<code>varchar(8),</code>
------------------	--------------------------

<i>proc_maj_ver</i>	<code>integer,</code>
---------------------	-----------------------

<i>proc_min_ver</i>	<code>integer,</code>
---------------------	-----------------------

<i>case_number</i>	<code>numeric(20))</code>
--------------------	---------------------------

where:

- *proc_name* is the name of the procedure that you want to purge a case of. The case must be either active or closed.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *case_num* is the number of the case that is to be purged.

Notes

Instead of using the specific major or minor version number or both of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

Example

This example purges case 103 of the CUSTREQ procedure.

```
EXEC owner.SW_PURGE 'CUSTREQ', -1, -1, 103
```

**Note**

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_PURGE_WITHOUT_EVENT

The SW_PURGE_WITHOUT_EVENT procedure purges the specified case of a procedure (permanently deleting it from the system) without triggering the events that are set for the OnBeforePurge event.

Syntax

```
SW_PURGE_WITHOUT_EVENT (
```

<i>proc_name</i>	<code>varchar(8),</code>
<i>proc_maj_ver</i>	<code>integer,</code>
<i>proc_min_ver</i>	<code>integer,</code>
<i>case_number</i>	<code>numeric(20))</code>

where:

- *proc_name* is the name of the procedure that you want to purge a case of. The case must be either active or closed.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *case_num* is the number of the case that is to be purged.

Notes

Instead of using the specific major or minor version number of the procedure, or both, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess uses the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).

**Note**

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

SW_SUSPEND

The SW_SUSPEND procedure suspends a case of a procedure.

Syntax

SW_SUSPEND (

<i>proc_name</i>	<code>varchar(8),</code>
------------------	--------------------------

<i>proc_maj_ver</i>	<code>integer,</code>
---------------------	-----------------------

<i>proc_min_ver</i>	<code>integer,</code>
---------------------	-----------------------

<i>case_number</i>	<code>numeric(20),</code>
--------------------	---------------------------

<i>user_id</i>	<code>varchar(24),</code>
----------------	---------------------------

<i>suspend_type</i>	<code>integer)</code>
---------------------	-----------------------

where:

- *proc_name* is the name of the procedure that you want to suspend a case of.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes.

- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes.
- *case_number* is the number of the case that is to be suspended.
- *user_id* is the name of the iProcess user who is suspending the case.
- *suspend_type* defines the type of suspend action. This 2.

Notes

Instead of using the specific major or minor version number or both of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

For more information about how to suspend and re-activate a case, please see the TIBCO iProcess Objects and TIBCO iProcess Server Objects programmer guide.

Example

This example suspends case 103 of the CUSTREQ procedure.

```
EXEC owner.SW_SUSPEND 'CUSTREQ', -1, -1, 103, 'swadmin', 2
```



Note

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

SW_ACTIVATE

The SW_ACTIVATE procedure re-activates a previously suspended case of a procedure.

Syntax

SW_ACTIVATE (

<i>proc_name</i>	<code>varchar(8),</code>
<i>proc_maj_ver</i>	<code>integer,</code>
<i>proc_min_ver</i>	<code>integer,</code>
<i>case_number</i>	<code>numeric(20),</code>
<i>user_id</i>	<code>varchar(24))</code>

where:

- *proc_name* is the name of the procedure that you want to reactivate a case of.
- *proc_maj_ver* is either the major version number of the *proc_name* procedure, or -1. See the notes .
- *proc_min_ver* is either the minor version number of the *proc_name* procedure, or -1. See the notes .
- *case_number* is the number of the suspended case that is to be reactivated.
- *user_id* is the name of the iProcess user who is re-activating the case.

Notes

Instead of using the specific major or minor version number or both of the procedure, you can specify both the *proc_maj_ver* and *proc_min_ver* parameters as -1. If you do this, iProcess will use the version number of the procedure that the case was originally started with or, that it has subsequently been migrated to (if a subsequent version has been released while the case is still in progress).



Note

If you specify one version number parameter as -1, you must specify the other one as -1 as well.

For more information about how to suspend and re-activate a case, please see the TIBCO iProcess Objects and TIBCO iProcess Server Objects programmer guide.

Example

This example re-activates case 103 of the CUSTREQ procedure.

```
EXEC owner.SW_ACTIVATE 'CUSTREQ', -1, -1, 103, 'swadmin'
```

**Note**

This example does not have any explicit transaction control, so will be committed immediately if using Transact-SQL.

Control Procedures

The following control procedures are available:

- [SW_ENABLECACHING](#)
- [SW_DISABLECACHING](#)
- [SW_SET_MBOX](#)
- [SW_SET_PRIORITY](#)
- [SW_SET_QUEUE](#)
- [SW_UNSET_MBOX](#)
- [SW_UNSET_PRIORITY](#)
- [SW_UNSET_QUEUE](#)
- [SW_INIT_TABLES](#)

SW_ENABLECACHING

The SW_ENABLECACHING procedure enables the caching of work item (reqid) and case number (casenum) sequence numbers for the current database session.

Syntax

```
SW_ENABLECACHING ( )
```

Notes

Caching reqid and casenum sequence numbers can be used to enhance batch SQL performance in appropriate situations.

When sequence number caching is enabled, the first transaction in the session retrieves its sequence numbers from the database, but subsequent transactions in the same session retrieve their sequences from the cache. (The size of the cache is set to 50 in the SW_ENABLECACHING procedure).

Unused sequence numbers in the cache are discarded when the database session terminates. This can result in gaps in the value of the sequence numbers if caching is used inappropriately. For example, if you enable caching for a session that simply starts a single case, all the unused iProcess case numbers will be lost.

Sequence number caching is enabled by default when a database session is started. Use the [SW_DISABLECACHING](#) procedure to disable sequence number caching.

For more information about sequence number caching, see "Sequence Number Caching" in *TIBCO iProcess Engine Administrator's Guide*.

SW_DISABLECACHING

The SW_DISABLECACHING procedure disables the caching of work item (reqid) and case number (casenum) sequence numbers for the current database session.

Syntax

```
SW_DISABLECACHING ()
```

Notes

Sequence number caching is enabled by default when a database session is started.

See the [SW_ENABLECACHING](#) procedure for more information about the use of sequence number caching, and when you should enable or disable it.

SW_SET_MBOX

The SW_SET_MBOX procedure tells the current SSOLite session to use a different Mbox set from the default one.

Syntax

```
SW_SET_MBOX (
  mbox_set_id    integer)
```

where:

- *mbox_set_id* is a unique identifier for the Mbox set you want to use.

Notes

This procedure is useful to partition messages for the purpose of performance or service levels. The procedure can be used in many ways, including for separating out bulk operations, such as purging or starting cases. Other sessions will still use the default Mbox set for operations such as delayed releases.

Use the [SW_UNSET_MBOX](#) procedure to restore using the default Mbox set for all operations.

Example

The following example shows how to set another Mbox set BGMBSETB for bulk operations. Remember to create the `sw_db_bgqueue_3` and `sw_db_bgqueue_4` physical queues and `bgmboxtable4` tables, and the `bgmboxqueue3` and `bgmboxqueue4` AQ first. For more information about queue processing and Mbox set creation, see [Processing Queues](#).

```
# Step 1. Add two new message queues.
#
swadm add_queue BGMBX3 Local 0003:swpro.sw_db_bgqueue_3
swadm add_queue BGMBX4 Local 0003:swpro.sw_db_bgqueue_4

# Step 2. Add a new Mbox set.
#
swadm add_mboxset BGMGSETB Local

# Step 3. View Mbox and queue IDs.
#
```

```

swadm show_mboxsets v
swadm show_queues

# Step 4. Add the BGMB0X3 and BGMB0X4 message queues to the BGMGSETB Mbox
set (
6 is the Mboxset ID of the BGMGSETB Mbox set,
8 is the queue ID of the BGMB0X3 message queue, and
9 is the queue ID of the BGMB0X4 message queue.)
#
swadm add_queue_to_mboxset 6 8
swadm add_queue_to_mboxset 6 9

# Step 5. Set the BGMGSETB Mbox set for bulk case starts.
#
EXEC swpro.SW_SET_MBOX 6

# Step 6. Start the bulk cases.
#
EXEC swpro.SW_ADD_PACK_DATA 'CustName', 'Allsop, J.A'
EXEC swpro.SW_ADD_PACK_DATA 'CustID', '478163'
EXEC swpro.SW_CASESTART 'CUSTREQ', -1, -1, 'Refund request', 'user35', '',
0, 0

# Step 7. Restore using the default Mbox set.
#
EXEC swpro.SW_UNSET_MBOX

```

SW_SET_PRIORITY

The SW_SET_PRIORITY procedure sets the internal message queue priorities. The procedure *only* changes the priority of the messages SSOLite sends. It does not change the SW_CP_VALUE and SW_IP_VALUE. So any subsequent messages for that case will remain at the default level or will be processed in the order of SW_CP_VALUE or SW_IP_VALUE when using iProcess Workspace (Windows) to process work items.

Syntax

```

SW_SET_PRIORITY (
    message_priority    integer)

```

where:

- *message_priority* is the priority value.

You can set priorities ranging from 1 to 999, where 1 is the highest priority, for internal message queues when passing messages between iProcess processes such as from the Background process to WIS processes, or from SSOLite to the Background process. Its default value is 50. See [Prioritizing Messages](#) for more information.

Notes

Use the [SW_UNSET_PRIORITY](#) procedure to restore the default message queue priorities.

Example

The following example sets the SW_CASESTART priority of Case1 and Case2 to 70, Case3 and Case4 to 100, and Case5 to the default priority.

```
begin
EXEC swpro.SW_SET_PRIORITY 70
EXEC swpro.SW_CASESTART 'CUSTREQ', -1, -1, 'Case1', 'user35', '', 0, 0
EXEC swpro.SW_CASESTART 'CUSTREQ', -1, -1, 'Case2', 'user35', '', 0, 0
EXEC swpro.SW_SET_PRIORITY 100
EXEC swpro.SW_CASESTART 'CUSTREQ', -1, -1, 'Case3', 'user35', '', 0, 0
EXEC swpro.SW_CASESTART 'CUSTREQ', -1, -1, 'Case4', 'user35', '', 0, 0
EXEC swpro.SW_UNSET_PRIORITY
EXEC swpro.SW_CASESTART 'CUSTREQ', -1, -1, 'Case5', 'user35', '', 0, 0
end
```

SW_SET_QUEUE

The SW_SET_QUEUE procedure forces all messages posted in the current database session to use the same background queue.

Syntax

```
SW_SET_QUEUE ()
```

Notes

By default, SSOLite stored procedures write messages to the BG processes using the default background message queues, using a round-robin allocation on a per-session basis. This allows the message load to be spread evenly across all of the available background queues. (See [Processing Queues](#) for more information.)

If required, you can use the `SW_SET_QUEUE` procedure to force all messages that are subsequently posted in the current session to use the same background queue.

After the `SW_SET_QUEUE` procedure has been called, the next message that is posted uses the next available background queue (as per normal round-robin allocation). Subsequent messages are then posted to the same queue, until either:

- the [SW_UNSET_QUEUE](#) procedure is called, after which messages are again allocated on the default round-robin basis, or
- the database session is terminated.

SW_UNSET_MBOX

The `SW_UNSET_MBOX` procedure restores using the default Mbox set for all operations.

Syntax

```
SW_UNSET_MBOX ()
```

Notes

Use the [SW_SET_MBOX](#) procedure to tell SSOLite to use a different Mbox set for bulk purges or bulk case starts.

Example

See the example of [SW_SET_MBOX](#).

SW_UNSET_PRIORITY

The SW_SET_PRIORITY procedure restores the default message queue priorities.

Syntax

```
SW_UNSET_PRIORITY ()
```

Note

You can set priorities for internal message queues when passing messages between iProcess processes such as from SSOLite to the BG process. See [Prioritizing Messages](#) for more information.

Use the [SW_SET_PRIORITY](#) procedure to set the internal message queue priorities.

SW_UNSET_QUEUE

The SW_SET_QUEUE procedure forces the use of round-robin queue allocation for messages posted in the current database session.

Syntax

```
SW_UNSET_QUEUE ()
```

Notes

The SW_UNSET_QUEUE procedure cancels the effect of a previous [SW_SET_QUEUE](#) procedure call. See the [SW_SET_QUEUE](#) procedure for more information.

SW_INIT_TABLES

The SW_INIT_TABLES procedure creates the temporary tables needed to store data for the current database session.

Syntax

```
SW_INIT_TABLES ()
```

Notes

The SW_INIT_TABLES procedure can be useful when a single database transaction is expected to run for a considerable period of time, because creating temporary tables during a transaction can block other SSOLite transactions.

You need only call SW_INIT_TABLES once per database session, although multiple calls will not affect performance.

Example

```
EXEC owner.SW_INIT_TABLES          -- Called outside of the transaction
-- to avoid unnecessary database
-- blocking.
begin transaction                  -- Start the transaction.
EXEC swpro.SW_CASESTART ...        -- Multiple SSOLite procedure calls.
EXEC ...
EXEC ...
.
.
commit                            -- Commit the transaction
```

Debug Procedures

The following debug procedures are available:

- [SW_SET_DEBUG](#)
- [SW_GET_DEBUG](#)
- [SW_CLEAR_DEBUG](#)

Debug output data is stored in the following temporary table:

```
##SSOLITE_DEBUG_DATA (
  spid      integer,
  message   varchar(255));
```

The table simply holds the debug message text in inserted order. If an application has enabled debugging, a simple `select * from SESSION.SSOLITE_DEBUG_DATA` statement can be used to display the debug data.


Note

Because this table is global you must store the database session id (`spid`) of the session that wrote this data. If multiple database sessions have debugging enabled, the data should be selected on the `spid`.

SW_SET_DEBUG

The `SW_SET_DEBUG` procedure turns debugging on or off.

Syntax

```
SW_SET_DEBUG(
  enable      integer)
```

where *enable* is a flag that turns debugging on or off. Specify:

- 1 to enable debugging (and create the `SSOLITE_DEBUG_DATA` temporary table for the session).
- 0 to disable debugging. Note that the `SESSION.SSOLITE_DEBUG_DATA` table is dropped and any existing data in the table lost.

SW_GET_DEBUG

`SW_GET_DEBUG` returns the number of rows of debug data available in the `SSOLITE_DEBUG_DATA` table, or -1 if debugging is not enabled.

Syntax

```
SW_GET_DEBUG() returns integer
```

Notes

Using SW_GET_DEBUG is optional.

SW_CLEAR_DEBUG

Calling SW_CLEAR_DEBUG clears all existing debug data and resets the SSOLITE_DEBUG_DATA temporary table.

Syntax

```
SW_CLEAR_DEBUG()
```

Notes

Use of this procedure is optional, as the use of temporary tables to hold debug data ensures that data is cleared anyway.

Database Stored Procedures

This appendix describes the package of database stored procedures.

Overview

Database stored procedures are available in the iProcess database and you can find them in the Oracle script file, `init2Kora_tok.sql`.

Sequence numbers can be generated by calling the stored procedures. See [Sequence Numbers](#) for more information.

The database stored procedures include:

- `sp_cdqp_cfg_sequence`
- `sp_cdqp_def_sequence`
- `sp_cnum_sequence`
- `sp_procid_sequence`
- `sp_reqid_sequence`
- `sp_waitid_sequence`
- `sp_iap_monitor_id_sequence`
- `sp_eaiws_jms_provider_seq`
- `sp_eaiws_jms_destination_seq`
- `casenum_find_gaps`

CASENUM_FIND_GAPS

The `CASENUM_FIND_GAPS` stored procedure adds a list of free case number gaps to the `casenum_gaps` table.

If the case number or the subcase number generated from the sequence table reaches the maximum case number, 4294967295, then the following cases cannot be started. This stored procedure is used to scan a range of case numbers and create available blocks of free case numbers for reuse. It operates across a case range and only allocates free case numbers. The free case numbers are available either because the case numbers have never been used or from the original cases that have been purged.

The `casenum_gaps` table is used to hold the free case number gaps that are created by the `CASENUM_FIND_GAPS` stored procedure. See [casenum_gaps](#) for more information.

Syntax

```
CASENUM_FIND_GAPS (
    v_casenum_min    IN NUMBER,
    v_casenum_max    IN NUMBER,
    v_gap_size       IN NUMBER)
```

where:

- `v_casenum_min` specifies the minimum case number of the range.
- `v_casenum_max` specifies the maximum case number of the range.
- `v_gap_size` specifies the minimum size of a gap that contains only free case numbers.

How to Reuse Free Case Numbers

Perform the following steps to reuse the free case numbers:



Note

TIBCO recommends that you shut down iProcess Engine before running `CASENUM_FIND_GAPS`. If you want to run the procedure against a running system, you must ensure that the case range supplied does not overlap with the ranges currently being used, as there is the possibility of overlapping gaps with duplicate case numbers being created.

1. Shut down TIBCO iProcess Engine.
2. Periodically run the `CASENUM_FIND_GAPS` stored procedure as the database administrator.

To do this, you can create a SQL script as shown in the following example, and use SQL*Plus to run the script.

```
exec swpro.casenum_find_gaps 1, 24, 1;
```

In the example, CASENUM_FIND_GAPS (100, 500, 20) looks for the gaps of at least 20 free case numbers from case number 100 to 500. If the range has three gaps: 130 - 140, 240 - 270, and 430 - 480, only the last two gaps will be listed in the casenum_gaps table for iProcess Engine to allocate case numbers.

3. Restart TIBCO iProcess Engine.

When TIBCO iProcess Engine wants to cache a new batch of sequences, it will first use the case numbers in the casenum_gaps table that are listed by the CASENUM_FIND_GAPS stored procedure and then allocate the unused new case numbers when the case numbers in the table are used up.

Notes

Before running the stored procedure, note that:

- Running the CASENUM_FIND_GAPS stored procedure may take a long time. It is only of benefit in the areas where the density of the occupied case numbers is low enough to have many gaps in between. This is typically in the lower range of case numbers, as these are older cases and more likely to have been closed and purged. The area close to the most recently started cases is likely to be densely populated, because all these cases are new and less likely to be closed and purged.
- It is recommended to have a good purge strategy to ensure that there are plenty of available case numbers for reuse.
- TIBCO recommends that you do not run CASENUM_FIND_GAPS repeatedly on the same case number range. Check the values in the casenum_gaps table for the listed gaps and run the procedure on a range outside of the highest and lowest figures in the table.
- The performance of CASENUM_FIND_GAPS is proportional not to the size of the range, but to the number of actual cases in the range. For instance, when running it on a range from 0 to 100 million, if there are only 5000 cases in that range, it will be very fast and might only take a few seconds. While running it on a range from 100 million to 105 million, if there are close to 5 million cases in that range, it will take considerably longer. To find how many cases are in the intended range, run the following SQL:

- `SELECT COUNT(*) FROM CASE_INFORMATION WHERE CASENUM > v_casenum_min
AND CASENUM < v_casenum_max`
- Based on previous runs and recorded timings, it should be possible to predict the time `CASENUM_FIND_GAPS` will take for any given range with a reasonable amount of accuracy.

See Also

- [casenum_gaps](#)

XPC_CREATE_INDEX

`XPC_CREATE_INDEX` is used for creating custom index on a given table based on user's need.

Syntax

```
XPC_CREATE_INDEX (
    idx_xpc_name    VARCHAR_TYPE(32))
```

The valid values for `idx_xpc_name` are:

- `idx_xpc_status`: Creates an index on case information table with columns `is_dead`, `casenum`, and `proc_id`.
- `idx_xpc_ended`: Creates an index on case information table with columns `next_deadline`, `casenum`, and `proc_id`.
- `idx_xpc_started`: Creates an index on case information table with columns `started`, `casenum`, and `proc_id`.
- `idx_xpc_starter`: Creates an index on case information table with columns `starter`, `casenum`, and `proc_id`.
- `idx_xpc_tid`: Creates an index on case information table with columns `type_id`, and `casenum`.
- `idx_xpc_date`: Creates an index on case information table with columns `audit_date`, and `casenum`.

- **idx_xpc_desc:** Creates an index on case information table with columns stepdesc, and casenum.
- **idx_xpc_step:** Creates an index on case information table with columns stepname, and casenum.
- **idx_xpc_user:** Creates an index on case information table with columns user_name, and casenum.

This stored procedure addresses any performance issues while searching millions of cases in the database. This logic facilitates iProcess Engine to check the filter and the tables involved. In the search, if there is no proper index created, the system throws a suggestion in sw_warn and system event to create index to improve the performance. The Engine check for the index on following fields, with the corresponding database table and database columns.

SYSTEM FIELD	INDEX INPUT	DB TABLE	DB COLUMN
SW_STATUS	idx_xpc_status	case_information	is_dead
SW_TERMINATED	idx_xpc_ended	case_information	next_deadline
SW_TERMINATEDDATE	idx_xpc_ended	case_information	next_deadline
SW_STARTED	idx_xpc_started	case_information	started
SW_STARTEDDATE	idx_xpc_started	case_information	started
SW_STARTER	idx_xpc_starter	case_information	starter
SW_AUD_TYPE	idx_xpc_tid	audit_trail	type_id
SW_AUD_DATE	idx_xpc_date	audit_trail	audit_date
SW_AUD_USER	idx_xpc_user	audit_trail	user_name
SW_AUD_STEP	idx_xpc_step	audit_trail	stepname
SW_AUD_DESC	idx_xpc_desc	audit_trail	stepdesc

This is primarily used when working with "Filtering Work Items and Cases" (Ref: TIBCO iProcess® Server Objects (Java) or (.NET) Programmer's Guide)

Unused Tables

The following tables are created by the database creation script (`init2Ksql.sql`), but are not currently used by the iProcess Engine:

- `pack_attach`
- `process_invqueue`
- `prounqid`
- `attachment`

**Warning**

Do not delete these tables. They are reserved for possible future use.

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [TIBCO Product Documentation](#) website, mainly in HTML and PDF formats.

The [TIBCO Product Documentation](#) website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

Documentation for TIBCO iProcess® Engine is available on the [TIBCO iProcess® Engine Product Documentation](#) page.

To directly access documentation for this product, double-click the following file:

`TIBCO_HOME/release_notes/TIB_ipe-sql_11.9.0_docinfo.html` where `TIBCO_HOME` is the top-level directory in which TIBCO products are installed. On Windows, the default `TIBCO_HOME` is `C:\tibco`. On UNIX systems, the default `TIBCO_HOME` is `/opt/tibco`.

The following documents for this product can be found in the TIBCO Documentation site:

- *TIBCO iProcess® Engine Architecture Guide*
- *TIBCO iProcess® Engine Configuration Guide for Cloud*
- TIBCO iProcess® Engine Administrator's Guides:
 - *TIBCO iProcess® Engine Administrator's Guide*
 - *TIBCO iProcess® Objects Director Administrator's Guide*
 - *TIBCO iProcess® Objects Server Administrator's Guide*
 - *TIBCO iProcess® Engine Administration Console User Guide*
- TIBCO iProcess® Engine Database Administrator's Guides:
 - *TIBCO iProcess® Engine (DB2) Administrator's Guide*

TIBCO iProcess® Engine (Oracle) Administrator's Guide

TIBCO iProcess® Engine (SQL) Administrator's Guide

TIBCO iProcess® swutil and swbatch Reference Guide

- *TIBCO iProcess® Engine System Messages Guide*
- *TIBCO iProcess® User Validation API User Guide*
- *LDAPCONF Utility User Guide*

How to Contact TIBCO Support

Get an overview of [TIBCO Support](#). You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the [TIBCO Support](#) website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to [TIBCO Support](#) website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

Legal and Third-Party Notices

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, ActiveMatrix BusinessWorks, Business Studio, Enterprise Message Service, Hawk, iProcess, and Rendezvous are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 1994-2022. TIBCO Software Inc. All Rights Reserved.