

# **TIBCO iProcess<sup>®</sup> Engine (DB2)**

## **Administrator's Guide**

*Software Release 11.4.1*

*April 2014*

## Important Information

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 contains confidential information that 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, Two-Second Advantage, TIBCO ActiveMatrix BusinessWorks, TIBCO Business Studio, TIBCO Enterprise Message Service, TIBCO Hawk, TIBCO iProcess, TIBCO iProcess Suite, and TIBCO Rendezvous are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

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.

Copyright © 1994-2014 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

# Contents

|   |               |
|---|---------------|
| <b>Preface</b> .....                                      | <b>ix</b>     |
| Related Documentation .....                               | x             |
| TIBCO iProcess Engine Documentation .....                 | x             |
| Other TIBCO Product Documentation .....                   | x             |
| Typographical Conventions .....                           | xii           |
| Connecting with TIBCO Resources .....                     | xv            |
| How to Join TIBCOCommunity .....                          | xv            |
| How to Access TIBCO Documentation .....                   | xv            |
| How to Contact TIBCO Support .....                        | xv            |
| <br><b>Chapter 1 The TIBCO iProcess Engine Node</b> ..... | <br><b>1</b>  |
| Table Relationships .....                                 | 2             |
| nodes .....   | 3             |
| <br><b>Chapter 2 Process Sentinels</b> .....              | <br><b>5</b>  |
| Table Relationships .....                                 | 6             |
| node_cluster .....  | 7             |
| process_config .....                                      | 9             |
| process_event_log .....                                   | 12            |
| process_attributes .....                                  | 14            |
| running_processes .....                                   | 16            |
| active_logins .....                                       | 18            |
| checksums .....   | 20            |
| sw_subscription_list .....                                | 21            |
| <br><b>Chapter 3 Mbox Sets and Message Queues</b> .....   | <br><b>23</b> |
| Table Relationships .....                                 | 24            |
| iql_queues .....  | 25            |
| mbox_set .....  | 28            |
| mbox_set_group .....                                      | 30            |
| Default DB2 Database Queue Tables (Test) .....            | 32            |
| sw_db_bgqueue_n .....                                     | 32            |
| sw_db_wisqueue_n .....                                    | 34            |

- sw\_db\_predictqueue\_n ..... 36
  - sw\_db\_deadqueue. .... 38
- Creating Additional DB2 Database Queue Tables ..... 40
  - Example. .... 40
- Chapter 4 Sequence Numbers ..... 43**
  - About Sequence Numbers ..... 44
  - Table Relationships ..... 46
  - sequences ..... 47
- Chapter 5 Procedures ..... 49**
  - Table Relationships ..... 50
  - proc\_index ..... 51
  - iap\_monitor ..... 55
  - iap\_field ..... 57
  - iap\_activity ..... 59
  - iap\_global ..... 61
  - proc\_version ..... 63
  - procedure\_lock ..... 66
  - proc\_instance ..... 68
  - proc\_audit ..... 70
  - proc\_defn ..... 72
  - proc\_deadline ..... 75
  - proc\_event ..... 77
  - wqd\_delta\_subscriptions ..... 80
- Chapter 6 Procedure Management ..... 81**
  - About Procedure Objects ..... 82
  - Table Relationships ..... 83
  - pm\_objects ..... 84
  - pm\_objects\_lock ..... 87
  - pmobjects\_security ..... 90
  - proc\_mgt\_hierarchy ..... 92
- Chapter 7 Cases ..... 95**
  - Table Relationships ..... 96
  - case\_information ..... 97

|   |            |
|---|------------|
| outstanding_addr .....                                      | 100        |
| wait .....  | 103        |
| wait_step .....   | 105        |
| status .....  | 107        |
| case_data .....   | 109        |
| audit_trail .....   | 111        |
| memo .....  | 114        |
| predict .....   | 116        |
| predict_lock .....  | 120        |
| case_deadline_event .....                                   | 122        |
| case_event .....  | 125        |
| casenum_gaps .....  | 128        |
| <b>Chapter 8 Work Items .....</b>                           | <b>131</b> |
| Table Relationships .....                                   | 132        |
| staffo .....  | 133        |
| pack_data .....   | 137        |
| pack_memo .....   | 139        |
| qaccess .....   | 141        |
| <b>Chapter 9 Case Data Queue Parameters .....</b>           | <b>143</b> |
| Table Relationships .....                                   | 144        |
| cdqp_def .....  | 145        |
| cdqp_cfg .....  | 147        |
| <b>Chapter 10 Queue Participation and Redirection .....</b> | <b>149</b> |
| Table Relationships .....                                   | 150        |
| part_defn .....   | 151        |
| part_list .....   | 153        |
| redir_defn .....  | 155        |
| <b>Chapter 11 Administrative Tables .....</b>               | <b>157</b> |
| Table Relationships .....                                   | 158        |
| flag_table .....  | 159        |
| version .....   | 162        |

**Chapter 12 Users and Work Queues. . . . . 163**

About User Tables. . . . . 164

Table Relationships. . . . . 165

user\_names . . . . . 166

user\_attrib . . . . . 168

user\_settings . . . . . 170

user\_values . . . . . 171

user\_memb . . . . . 173

leavers . . . . . 175

tsys\_user\_names . . . . . 177

tsys\_user\_attrib . . . . . 178

tsys\_user\_values . . . . . 179

tsys\_user\_memb . . . . . 180

**Chapter 13 Roles . . . . . 181**

About Roles . . . . . 182

Table Relationships. . . . . 183

role\_users . . . . . 184

tsys\_role\_users . . . . . 186

**Chapter 14 TIBCO iProcess Tables. . . . . 187**

About TIBCO iProcess Tables. . . . . 188

Table Relationships. . . . . 189

db\_names . . . . . 190

db\_fields . . . . . 192

db\_values . . . . . 194

tsys\_db\_names . . . . . 196

tsys\_db\_fields . . . . . 197

tsys\_db\_values . . . . . 198

str\_db\_names . . . . . 199

str\_db\_fields . . . . . 200

ttmp\_db\_names . . . . . 201

ttmp\_db\_fields . . . . . 202

ttmp\_db\_values . . . . . 203

|   |            |
|---|------------|
| <b>Chapter 15 Lists</b>                     | <b>205</b> |
| About Lists                                 | 206        |
| Table Relationships                         | 207        |
| list_names                                  | 208        |
| list_values                                 | 210        |
| tsys_list_names                             | 212        |
| tsys_list_values                            | 213        |
| ttmp_list_names                             | 214        |
| ttmp_list_values                            | 215        |
| <b>Chapter 16 iProcess Server Plug-ins</b>  | <b>217</b> |
| Table Relationships                         | 218        |
| eai_registry                                | 219        |
| <b>Chapter 17 Firewall Port Ranges</b>      | <b>221</b> |
| Table Relationships                         | 222        |
| port_range                                  | 223        |
| port_range_active                           | 225        |
| port_range_conf                             | 227        |
| port_range_nodes                            | 229        |
| <b>Chapter 18 WQS/WIS Shared Memory</b>     | <b>231</b> |
| Table Relationships                         | 232        |
| wqs_index                                   | 233        |
| <b>Appendix A Views</b>                     | <b>237</b> |
| <b>Appendix B SSOLite Stored Procedures</b> | <b>239</b> |
| Overview                                    | 240        |
| Using SSOLite Stored Procedures             | 241        |
| Processing Asynchronous Message             | 241        |
| Transactional Processing                    | 241        |
| Post-installation Procedures                | 242        |
| Handling Exceptions                         | 242        |
| Processing Queues                           | 244        |
| Prioritizing Messages                       | 245        |
| Data Procedures                             | 247        |
| SW_ADD_PACK_DATA                            | 248        |

- SW\_ADD\_PACK\_MEMO ..... 250
- SW\_CLEAR\_PACK\_CACHE ..... 251
- SW\_MODIFY\_CASEDATA..... 252
- Command Procedures ..... 254
  - SW\_AUDIT ..... 255
  - SW\_CASEREOPEN ..... 257
  - SW\_CASESTART ..... 258
  - SW\_CLOSE..... 260
  - SW\_CLOSE\_WITHOUT\_EVENT ..... 261
  - SW\_EVENT..... 262
  - SW\_EVENT\_UPDATE\_PACK ..... 263
  - SW\_GETCASE\_STATUS..... 264
  - SW\_GRAFT..... 265
  - SW\_GRAFTCOUNT ..... 267
  - SW\_JUMPTO ..... 269
  - SW\_JUMPTO\_MULTI ..... 271
  - SW\_PURGE ..... 273
  - SW\_PURGE\_WITHOUT\_EVENT ..... 274
  - SW\_SUSPEND ..... 275
  - SW\_ACTIVATE ..... 276
- Control Procedures..... 277
  - SW\_ENABLECACHING ..... 278
  - SW\_DISABLECACHING ..... 279
  - SW\_SET\_MBOX ..... 280
  - SW\_SET\_PRIORITY..... 282
  - SW\_SET\_QUEUE ..... 283
  - SW\_UNSET\_MBOX..... 284
  - SW\_UNSET\_PRIORITY ..... 285
  - SW\_UNSET\_QUEUE ..... 286
- Debug Procedures ..... 287
  - SW\_SET\_DEBUG ..... 288
  - SW\_GET\_DEBUG..... 289
  - SW\_CLEAR\_DEBUG ..... 290
- Appendix C Database Stored Procedures ..... 291**
  - Overview..... 292
  - CASENUM\_FIND\_GAPS ..... 293

- Appendix D Unused Tables ..... 297**

# Preface

This guide describes the TIBCO iProcess Engine (DB2) database schema.

## Topics

---

- [Related Documentation, page x](#)
- [Typographical Conventions, page xii](#)
- [Connecting with TIBCO Resources, page xv](#)

## Related Documentation

---

This section lists documentation resources you may find useful.

### TIBCO iProcess Engine Documentation

The following documents form the TIBCO iProcess Engine documentation set:

- *TIBCO iProcess Engine Installation* Read this manual for instructions on site preparation and installation.
- *TIBCO iProcess Engine Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.
- **TIBCO iProcess Suite Documentation** This documentation set contains all the manuals for TIBCO iProcess Engine and other TIBCO products in TIBCO iProcess® Suite. The manuals for TIBCO iProcess Engine are as follows:
  - *TIBCO iProcess Engine Architecture Guide*
  - **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 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's Guide*
  - *LDAPCONF Utility User's Guide*

### Other TIBCO Product Documentation

You may find it useful to read the documentation for the following TIBCO products:

- TIBCO ActiveMatrix BusinessWorks™

- TIBCO Business Studio™
- TIBCO Enterprise Message Service™
- TIBCO Hawk®
- TIBCO Rendezvous®

# Typographical Conventions

TIBCO iProcess Engine can be run on both Microsoft Windows and UNIX/Linux platforms. In this manual, the Windows convention of a backslash (\) is used. The equivalent pathname on a UNIX or Linux system is the same, but using the forward slash (/) as a separator character.



UNIX or Linux pathnames are occasionally shown explicitly, using forward slashes as separators, where a UNIX or Linux-specific example or syntax is required.

Any references to UNIX in this manual also apply to Linux unless explicitly stated otherwise.

The following typographical conventions are used in this manual

Table 1 General Typographical Conventions

| Convention   | Use  |
|--------------|--|
| <i>SWDIR</i> | <p>TIBCO iProcess Engine installs into a directory. This directory is referenced in documentation as <i>SWDIR</i>. The value of <i>SWDIR</i> depends on the operating system. For example,</p> <ul style="list-style-type: none"><li>on a Windows server (on the C: drive)<br/>if <i>SWDIR</i> is set to the C:\swserver\staffw_nod1 directory, then the full path to the <code>swutil</code> command is in the C:\swserver\staffw_nod1\bin\swutil directory.</li><li>on a UNIX or Linux server<br/>if <i>SWDIR</i> is set to the /swserver/staffw_nod1 directory, then the full path to the <code>swutil</code> command is in the /swserver/staffw_nod1/bin/swutil directory or the \$<i>SWDIR</i>/bin/swutil directory.</li></ul> <p><b>Note:</b> On a UNIX or Linux system, the environment variable \$<i>SWDIR</i> should be set to point to the iProcess system directory for the <i>root</i> and <i>swadmin</i> users.</p> |
| code font    | <p>Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example:</p> <p>Use <code>MyCommand</code> to start the <code>foo</code> process.</p>   |

Table 1 General Typographical Conventions (Cont'd)




| Convention  | Use  |
|---|--|
| <b>bold code font</b>   | <p>Bold code font is used in the following ways:</p> <ul style="list-style-type: none"> <li>• In procedures, to indicate what a user types. For example: Type <b>admin</b>.</li> <li>• In large code samples, to indicate the parts of the sample that are of particular interest.</li> <li>• In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, MyCommand is enabled:<br/>MyCommand [<b>enable</b>   disable]</li> </ul>      |
| <i>italic font</i>  | <p>Italic font is used in the following ways:</p> <ul style="list-style-type: none"> <li>• To indicate a document title. For example: See <i>TIBCO ActiveMatrix BusinessWorks Concepts</i>.</li> <li>• To introduce new terms. For example: A portal page may contain several portlets. <i>Portlets</i> are mini-applications that run in a portal.</li> <li>• To indicate a variable in a command or code syntax that you must replace. For example: MyCommand <i>PathName</i></li> </ul> |
| Key combinations  | <p>Key name separated by a plus sign indicate keys pressed simultaneously. For example: Ctrl+C.</p> <p>Key names separated by a comma and space indicate keys pressed one after the other. For example: Esc, Ctrl+Q.</p>   |
|  | The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.   |
|  | The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result.   |
|  | The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken.   |

Table 2 Syntax Typographical Conventions

| Convention | Use   |
|------------|---|
| [ ]        | <p>An optional item in a command or code syntax.</p> <p>For example:</p> <p>MyCommand [optional_parameter] required_parameter</p> |

Table 2 Syntax Typographical Conventions (Cont'd)

| Convention | Use   |
|------------|---|
|            | <p>A logical OR that separates multiple items of which only one may be chosen.</p> <p>For example, you can select only one of the following parameters:</p> <p>MyCommand param1   param2   param3</p>   |
| { }        | <p>A logical group of items in a command. Other syntax notations may appear within each logical group.</p> <p>For example, the following command requires two parameters, which can be either the pair param1 and param2, or the pair param3 and param4.</p> <p>MyCommand {param1 param2}   {param3 param4}</p> <p>In the next example, the command requires two parameters. The first parameter can be either param1 or param2 and the second can be either param3 or param4:</p> <p>MyCommand {param1   param2} {param3   param4}</p> <p>In the next example, the command can accept either two or three parameters. The first parameter must be param1. You can optionally include param2 as the second parameter. And the last parameter is either param3 or param4.</p> <p>MyCommand param1 [param2] {param3   param4}</p> |

## Connecting with TIBCO Resources

---

### How to Join TIBCOCommunity

TIBCOCommunity is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCOCommunity offers forums, blogs, and access to a variety of resources. To register, go to <http://www.tibcommunity.com>.

### How to Access TIBCO Documentation

You can access TIBCO documentation here:

<http://docs.tibco.com>

### How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support as follows:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

<http://www.tibco.com/services/support>

- If you already have a valid maintenance or support contract, visit this site:

<https://support.tibco.com>

Entry to this site requires a user name and password. If you do not have a user name, you can request one.



## Chapter 1

# The TIBCO iProcess Engine Node

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

## Topics

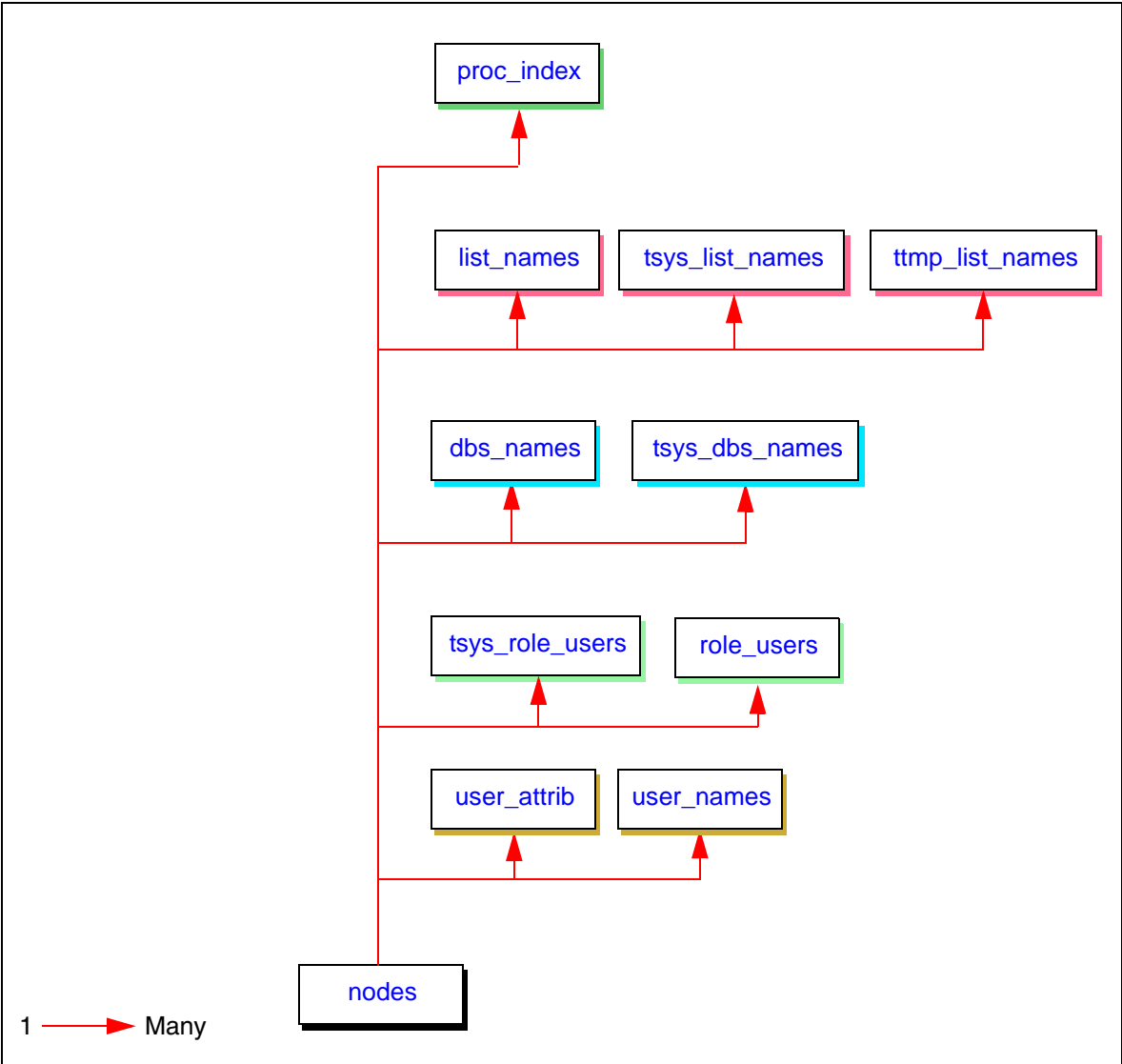
---

- [Table Relationships, page 2](#)
- [nodes, page 3](#)

# Table Relationships

The following diagram shows how the **nodes** table is related 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.



## 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)     ,
  mail_cert        VARCHAR(31)     ,
  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.<br>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 (SWDIR).        |
| 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>                |

| Column         | Description  |
|----------------|--|
| 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   |

**Foreign Keys**      None.

**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.  |

## Chapter 2      **Process Sentinels**

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

### Topics

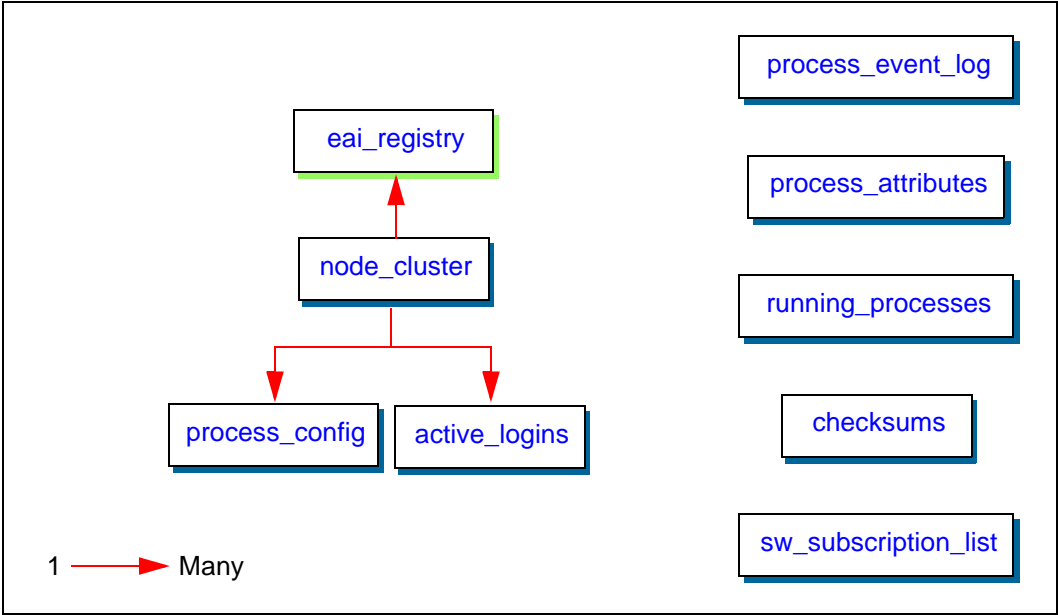
---

- [Table Relationships, page 6](#)
- [node\\_cluster, page 7](#)
- [process\\_config, page 9](#)
- [process\\_event\\_log, page 12](#)
- [process\\_attributes, page 14](#)
- [running\\_processes, page 16](#)
- [active\\_logins, page 18](#)
- [checksums, page 20](#)
- [sw\\_subscription\\_list, page 21](#)

# Table Relationships

The following diagram shows how the tables described in this chapter 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.



## 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(
  logical_machine_id    INTEGER                NOT NULL,
  physical_machine_name VARCHAR(256)           NOT NULL,
  master                SMALLINT              NOT NULL,
  check_error_files     SMALLINT              NOT NULL,
  machine_comment       VARCHAR(256)           )
```

| Column                             | Description   |
|------------------------------------|---|
| <code>logical_machine_id</code>    | Unique ID for this server.  |
| <code>physical_machine_name</code> | The name of this server (as returned by the UNIX <code>uname</code> command).   |
| <code>master</code>                | 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).                               |
| <code>check_error_files</code>     | Flag that defines whether the Process Sentinels on this server check (1) or do not check (0) for the creation of <code>SWDIR\logs\sw_error</code> and <code>sw_warn</code> files. |
| <code>machine_comment</code>       | Descriptive comment describing this server.   |

**Primary Key** The following primary key is defined for this table.

| Key Name                     | Column(s)                       |
|------------------------------|---------------------------------|
| <code>pk_node_cluster</code> | <code>logical_machine_id</code> |

**Foreign Keys** None.

**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 <i>SWDIR\util\swadm</i> utility. |
| updated     | a server's details are updated, using the <i>SWDIR\util\swadm</i> utility.                                 |
| deleted     | a server is removed from the node, using the <i>SWDIR\util\swadm</i> 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)    )
```

| Column                                | Description  |
|---------------------------------------|--|
| <code>logical_machine_id</code>       | ID of the server where this process instance runs, as defined in the <a href="#">node_cluster</a> table.   |
| <code>logical_process_name</code>     | Logical name of this process instance.<br><b>Note:</b> 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> | Unique ID for this process instance.   |
| <code>enabled</code>                  | Flag that defines whether this process instance starts automatically (1) when the iProcess Engine starts, or whether it must be started manually (0).  |
| <code>persistent</code>               | Flag that defines whether this process instance is automatically restarted (1) or not (0) when the iProcess Engine is shut down and restarted.<br><b>Note:</b> Any row in which the persistent value is 0 is deleted when the iProcess Engine starts up. |

| Column            | Description   |
|-------------------|---|
| last_known_status | Last known status of this process instance, as reported to the Process Sentinels by the process.<br><br>Either: STARTING, RUNNING, PAUSED, SUSPENDED, SHUTTING DOWN or STOPPED.<br><br><b>Note:</b> The <a href="#">process_event_log</a> table provides an audit trail of changes to the status of a process instance. |
| 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<br>logical_process_name<br>logical_process_instance |

**Foreign Keys**      The following foreign key is defined for this table.

| Key Name          | Column(s)          | Referenced in Table...       |
|-------------------|--------------------|------------------------------|
| fk_process_config | logical_machine_id | <a href="#">node_cluster</a> |

**Indexes**      The following index is defined for this table.

| Index Name         | Column(s) Indexed  |
|--------------------|--------------------|
| idx_processconf_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. |

| A row is... | When...  |
|-------------|--|
| updated     | a process instance's settings or status are updated, either by system activity, or by using the <i>SWDIR\util\swadm</i> , <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> , <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             TIMESTAMP             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 <a href="#">node_cluster</a> table.  |
| logical_process_name     | Logical name of the process that this event applies to.<br><b>Note:</b> See "Administering iProcess Engine Server Processe" 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 <a href="#">process_config</a> 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"><li>• 3000 - process instance started.</li><li>• 3001 - process instance stopping.</li><li>• 3002 - process instance stopped.</li><li>• 3003 - process instance died.</li><li>• 3004 - process instance paused.</li><li>• 3005 - process instance unpaused.</li></ul> |
| 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.   |

|                       |   |
|-----------------------|---|
| <b>Primary Key</b>    | None.   |
| <b>Foreign Keys</b>   | None.   |
| <b>Indexes</b>        | None.   |
| <b>Table Activity</b> | <p>The <code>process_event_log</code> table contains one row for each status change event that has occurred to each instance of a server process.</p> <p>Rows are added, updated and deleted in the following situations.</p> |

| A row is... | When...  |
|-------------|--|
| added       | a process starts, receives a shutdown command, or shuts down.  |
| updated     | never.   |
| deleted     | never.<br><b>Note:</b> 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_machine_id       | ID of the server where the process instance that this attribute applies to is running, as defined in the <a href="#">node_cluster</a> table.<br><br>A value of 0 means that this attribute applies to all servers that are part of this node.   |
| logical_process_name     | Logical name of the process that this attribute applies to.<br><br>A value of ALL means that this attribute applies to all processes on the indicated server.<br><br><b>Note:</b> 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 the process instance that this attribute applies to, as defined in the <a href="#">process_config</a> table.<br><br>A value of 0 means that this attribute applies to all instances of the indicated process.   |
| attribute_name           | Name of this process attribute.<br><br><b>Note:</b> See "Administering Process Attributes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of the available process attributes.  |
| attribute_values         | Value of this process attribute.<br><br><b>Note:</b> See "Administering Process Attributes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of the valid values for each process attributes.   |

| Column         | Description   |
|----------------|---|
| attribute_type | Type of this process attribute: either I (Integer), C (Character) or S (String).<br><br><b>Note:</b> 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. |

**Primary Key** The following primary key is defined for this table.

| Key Name              | Column(s)  |
|-----------------------|--|
| pk_process_attributes | logical_machine_id<br>logical_process_name<br>logical_process_instance<br>attribute_name |

**Foreign Keys** 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.                                |



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  |
|--------------------------|--|
| logical_machine_id       | ID of the server where this process instance is running, as defined in the <a href="#">node_cluster</a> table.   |
| logical_process_name     | Logical name of this process instance.<br><b>Note:</b> 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 <a href="#">process_config</a> table.   |
| process_id               | Process ID (PID) of this process instance.   |
| port_number              | Port number that this process instance is running on.  |

**Primary Key** None.

**Foreign Keys** None.

**Indexes** The following index is defined for this table.

| Index Name         | Column(s) Indexed  |
|--------------------|--|
| idx_runningproc_fk | logical_machine_id<br>logical_process_name<br>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 <a href="#">node_cluster</a> table.   |
| logical_process_name     | Logical name of this process instance.<br><b>Note:</b> 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 <a href="#">process_config</a> table.   |
| user_name                | Name of the user who is logged in, as defined in the <a href="#">user_names</a> 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.

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                      | Column(s)          | Referenced in Table...       |
|-------------------------------|--------------------|------------------------------|
| fk_active_logins <sup>1</sup> | logical_machine_id | <a href="#">node_cluster</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following indexes are defined for this table.

| Index Name         | Column(s) Indexed  |
|--------------------|--------------------|
| idx_activelogns_fk | logical_machine_id |
| idx_active_logins  | 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 Enginee to provide security checks on the [active\\_logins](#) and [port\\_range](#) tables.

**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   |

**Foreign Keys** None.

**Indexes** None.

**Table Activity** The checksums table contains one row for each checksum used internally bythe 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.                          |

## sw\_subscription\_list

The `sw_subscription_list` table stores the list of *event subscription* records currently defined on the system. Each event subscription record defines the event that is being subscribed to and the process instance that has subscribed to it.

The `sw_subscription_list` table is a persistent copy of the event daemon's in-memory subscription list. If the event daemon fails for any reason, it rebuilds its subscription list from this table when it restarts.

**Structure** The `sw_subscription_list` table has the following structure:

```
TABLE sw_subscription_list (
  event_topic_id      integer          NOT NULL,
  logical_machine_id  integer          NOT NULL,
  logical_process_name varchar(10)     NOT NULL,
  logical_process_instance integer      NOT NULL,
  notify_port_number  integer          NOT NULL,
  machine_name        varchar(256)     NOT NULL)
```

| Column                                | Description   |
|---------------------------------------|---|
| <code>event_topic_id</code>           | ID of the iProcess event that has been subscribed to.   |
| <code>logical_machine_id</code>       | ID of the server where the subscribing process is running, as defined in the <a href="#">node_cluster</a> table.  |
| <code>logical_process_name</code>     | Logical name of the process that has subscribed to this event.<br><b>Note:</b> 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 the process instance that has subscribed to this event, as defined in the <a href="#">process_config</a> table.   |
| <code>notify_port_number</code>       | Number of the RPC port that is used to notify the subscribing process when the event occurs.  |
| <code>machine_name</code>             | Physical name of the server where the subscribing process is running, as defined in the <a href="#">node_cluster</a> table.   |

**Primary Key**      The following primary key is defined for this table.

| Key Name       | Column(s)  |
|----------------|--|
| pk_sw_sub_list | event_topic_id<br>logical_machine_id<br>logical_process_name<br>logical_process_instance |

**Foreign Keys**      None.

**Indexes**      None.

**Table Activity**      The sw\_subscription\_list table contains one row for each event subscription currently defined on the system.

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

| A row is... | When...  |
|-------------|--|
| added       | a process instance subscribes to an event.   |
| updated     | details of a subscribing process are changed.<br>For example, if a process in a node cluster is moved to another server. |
| deleted     | a process instance unsubscribes from an event.   |

## Chapter 3

# Mbox Sets and Message Queues

This chapter describes the tables that are used to control the behavior of the message queueing system.

It also describes the [Default DB2 Database Queue Tables \(Test\)](#) which provide the underlying message queueing system used by the iProcess message queues.

## Topics

---

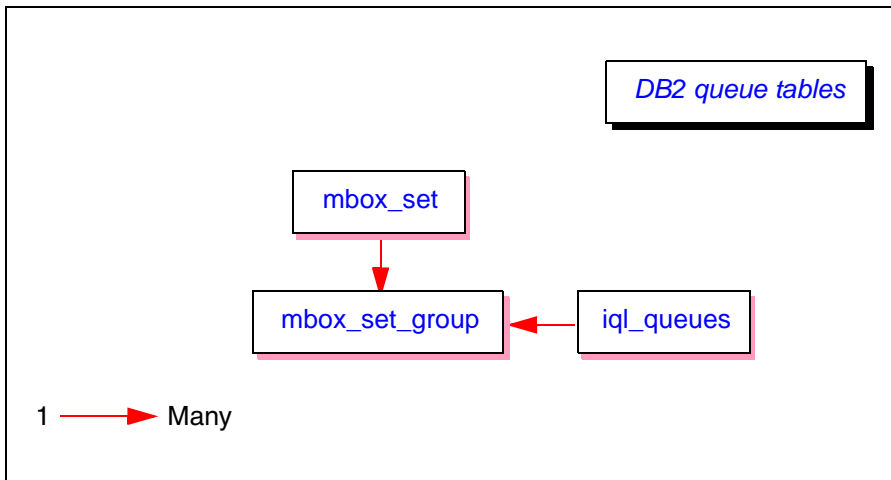
- [Table Relationships](#), page 24
- [iql\\_queues](#), page 25
- [mbox\\_set](#), page 28
- [mbox\\_set\\_group](#), page 30
- [Default DB2 Database Queue Tables \(Test\)](#), page 32
- [Creating Additional DB2 Database Queue Tables](#), page 40

## Table Relationships

---

The following diagram shows how the tables described in this chapter 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.



## 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 DB2 database queue table that is used to hold this message queue. See:<br><a href="#">Format of the DB2 Database Queue TableID on page 25</a> for a description of the format used for this value.<br><a href="#">Default DB2 Database Queue Tables (Test) on page 32</a> for more information about the default SQL database queue tables, and how to create additional tables. |

### Format of the DB2 Database Queue TableID

The ID of the DB2 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 DB2 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.
- *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 DB2 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 DB2 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 DB2 database table called `sw_db_bgqueue_3`, which is stored in the `sw` database (on the DB2 Server hosting the iProcess database) and owned by user `swpro1`.

Default Message Queues and Tables

When the iProcess Engine is installed, the `init2Kdb2.sql` script creates the following default set of message queues and DB2 database queue tables required by the system.

| Queue Name   | DB2 Database Table Queue ID     |
|--------------|---------------------------------|
| BGMBOX1      | 0003:swpro.sw_db_bgqueue_1      |
| BGMBOX2      | 0003:swpro.sw_db_bgqueue_2      |
| WISMBOX1     | 0003:swpro.sw_db_wisqueue_1     |
| WISMBOX2     | 0003:swpro.sw_db_wisqueue_2     |
| DEADQUEUE    | 0003:swpro.sw_db_deadqueue      |
| PREDICTMBOX1 | 0003:swpro.sw_db_predictqueue_1 |
| PREDICTMBOX2 | 0003:swpro.sw_db_predictqueue_2 |

**Primary Key** The following primary key is defined for this table.

| Key Name      | Column(s) |
|---------------|-----------|
| pk_iql_queues | queue_id  |

**Foreign Keys** 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. |
| 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   |
|------------------|---|
| mbox_set_id      | Unique identifier for this Mbox set.  |
| mbox_set_name    | Name of this Mbox set.  |
| mbox_set_msgtype | 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 `init2Kdb2.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 |

**Foreign Keys** None.

**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 <code>SWDIR\util\swadm</code> utility. |
| updated     | an Mbox set's details are updated, using the <code>SWDIR\util\swadm</code> utility.                                |
| deleted     | an Mbox set is deleted from the node, using the <code>SWDIR\util\swadm</code> 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   |
|---------------|---|
| mbox_set_id   | Unique identifier of the Mbox set that contains the associated message queue, as defined in the <a href="#">mbox_set</a> table.         |
| mbox_queue_id | Unique identifier of the message queue that is included in the associated Mbox set, as defined in the <a href="#">iql_queues</a> table. |

**Primary Key** The following primary key is defined for this table.

| Key Name          | Column(s)                    |
|-------------------|------------------------------|
| pk_mbox_set_group | mbox_set_id<br>mbox_queue_id |

**Foreign Keys** The following foreign keys are defined for this table.

| Key Name                    | Column(s)   | Referenced in Table...     |
|-----------------------------|-------------|----------------------------|
| fk_mbox_set_id <sup>1</sup> | mbox_set_id | <a href="#">mbox_set</a>   |
| fk_mbox_queue_id            | queue_id    | <a href="#">iql_queues</a> |

1. This key enforces the `DELETE CASCADE` referential action.

**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 DB2 Database Queue Tables (Test)

Each message queue defined in the [iql\\_queues](#) table must be mapped to its own DB2 database queue table.

When the iProcess Engine is installed, the `init2Kdb2.sql` script creates the default set of queue tables required by the system (see [Default Message Queues and Tables on page 26](#)).

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

| Queue Table                                  | See                                  |
|--|--------------------------------------|
| sw_db_bgqueue_1<br>sw_db_bgqueue_2           | <a href="#">sw_db_bgqueue_n</a>      |
| sw_db_wisqueue_1<br>sw_db_wisqueue_2         | <a href="#">sw_db_wisqueue_n</a>     |
| sw_db_predictqueue_1<br>sw_db_predictqueue_2 | <a href="#">sw_db_predictqueue_n</a> |
| sw_db_deadqueue                              | <a href="#">sw_db_deadqueue</a>      |

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 DB2 Database Queue Tables on page 40](#) 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)          NOT NULL  
GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1,  
    CACHE 50),  
    last_failed    numeric(10)          ,  
    failure_count  integer              NOT NULL,  
    msg_id         varchar(16)         FOR BIT DATA NOT
```

|          |               |           |
|----------|---------------|-----------|
| NULL,    |               |           |
| msg_hdr  | varchar(500)  | ,         |
| msg_data | varchar(1000) | 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.<br><br>When this value equals or exceeds the value of the IQL_RETRY_DELAY process attribute, the message is retried.           |
| failure_count | Number of times that this message has failed to be processed.<br><br>When this value equals or exceeds the value of the IQL_RETRY_COUNT process attribute, the message is moved to the <a href="#">sw_db_deadqueue</a> . |
| msg_id        | Unique identifier of this message.   |
| 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.<br><br>The default value is 50.   |



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.

**Indexes** The following index is defined for this table.

| Index Name                 | Indexed Column(s) |
|----------------------------|-------------------|
| idx_bgqueue_n <sup>1</sup> | row_id            |

1. UNIQUE index

**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:<br>successfully processes it.<br>cannot successfully process it, and moves it to the <a href="#">sw_db_deadqueue</a> 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)                NOT NULL  
GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1,  
CACHE 50),  
    last_failed          numeric(10)                ,  
    failure_count        integer                    NOT NULL,  
    msg_id               varchar(16) FOR BIT DATA NOT NULL,  
    msg_hdr              varchar(500)                ,  
    msg_data             varchar(1000)               NOT NULL  
    priority             integer                    NOT NULL)
```

| Column | Description  |
|--------|--|
| rowid  | Identifier of the row in the table for this message. |

| Column                     | Description  |
|----------------------------|--|
| <code>last_failed</code>   | Number of seconds since January 1st, 1970, when this message last failed to be processed.<br>When this value equals or exceeds the value of the <code>IQL_RETRY_DELAY</code> process attribute, the message is retried.        |
| <code>failure_count</code> | Number of times that this message has failed to be processed.<br>When this value equals or exceeds the value of the <code>IQL_RETRY_COUNT</code> process attribute, the message is moved to the <code>sw_db_deadqueue</code> . |
| <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>      | Message queue priority. The lower this value is, the higher the message queue priority is.<br>The default value is 50.   |



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.

**Indexes** The following index is defined for this table.

| Index Name                               | Indexed Column(s)   |
|--|---------------------|
| <code>idx_wisqueue_n</code> <sup>1</sup> | <code>row_id</code> |

1. UNIQUE index

**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. |

| A row is... | When...  |
|-------------|--|
| updated     | the WISMBD process dequeues a message from this table but cannot successfully process it (but the message has not exceeded the IQL_RETRY_COUNT value).   |
| deleted     | the WISMBD process dequeues a message from this table and either: successfully processes it.<br>cannot successfully process it, and moves it to the <a href="#">sw_db_deadqueue</a> table because it has exceeded the IQL_RETRY_COUNT 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:

```
TABLE sw_db_predictqueue_n (  
  rowid                numeric(15)          NOT NULL  
  GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1,  
  CACHE 50),  
  last_failed          numeric(10)           ,  
  failure_count        integer              NOT NULL,  
  msg_id               varchar(16) FOR BIT DATA NOT NULL,  
  msg_hdr              varchar(500)         ,  
  msg_data             varchar(1000)        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.<br><br>When this value equals or exceeds the value of the IQL_RETRY_DELAY process attribute, the message is retried. |

| Column        | Description  |
|---------------|--|
| failure_count | Number of times that this message has failed to be processed.<br>When this value equals or exceeds the value of the IQL_RETRY_COUNT process attribute, the message is moved to the <a href="#">sw_db_deadqueue</a> . |
| msg_id        | Unique identifier of this message.   |
| 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.<br>The default value is 50.   |



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.

**Indexes** The following index is defined for this table.

| Index Name                  | Indexed Column(s) |
|-----------------------------|-------------------|
| idx_predictq_n <sup>1</sup> | row_id            |

1. UNIQUE index

**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). |

| A row is... | When...  |
|-------------|--|
| deleted     | a BGPREDICT process dequeues a message from this table and either:<br>successfully processes it.<br>cannot successfully process it, and moves it to the <a href="#">sw_db_deadqueue</a> table because it has exceeded the IQL_RETRY_COUNT value. |

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(128)          NOT NULL,  
    rowid              numeric(15)            NOT NULL  
    GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),  
    last_failed        numeric(10)            ,  
    failure_count      integer                NOT NULL,  
    msg_id             varchar(16) FOR BIT DATA NOT NULL,  
    msg_hdr            varchar(500)           NULL,  
    msg_data           varchar(1000)          NOT NULL  
    priority           integer                NOT NULL)
```

| Column        | Description   |
|---------------|---|
| failed_by     | Identifies the queue table that this message originates from. One of the following processes:<br>BG (for a message from a <a href="#">sw_db_bgqueue_n</a> table).<br>WIS (for a message from a <a href="#">sw_db_wisqueue_n</a> table).<br>BGPREDICT (for a message from a <a href="#">sw_db_predictqueue_n</a> 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.<br><b>Note:</b> 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.<br>The default value is 50. |

**Primary Key** No primary key is defined for this table.

**Indexes** The following index is defined for this table.

| Index Name                 | Indexed Column(s) |
|----------------------------|-------------------|
| idx_deadqueue <sup>1</sup> | row_id            |

1. UNIQUE index

**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.

| A row is... | When...  |
|-------------|--|
| added       | a message is moved to this table from a <a href="#">sw_db_bgqueue_n</a> , <a href="#">sw_db_wisqueue_n</a> or <a href="#">sw_db_predictqueue_n</a> table, because it has exceeded the IQL_RETRY_COUNT value. |
| updated     | never.   |
| deleted     | never.   |

## Creating Additional DB2 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.
- 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.



If a table that does not conform to these requirements is used as a message queue, messages will not be able to 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 are no longer able to cope. To deal with the additional load you have decided that you need to add a new BGMBX3 message queue to the BGMBSET Mboxset. This queue requires a new DB2 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 DB2 Server that holds the iProcess database.
2. Create a new table in the iProcess database called `sw_db_bgqueue_3`.

For example:

```
CREATE TABLE swpro.sw_db_bgqueue_3 (
rowid          numeric(15) NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH
1, INCREMENT BY 1),
last_failed    numeric(10) ,
failure_count  integer      NOT NULL,
msg_id         varchar(16) FOR BIT DATANOT NULL,
msg_hdr        varchar(500) ,
msg_data       varchar(1000)NOT NULL)/
```

```
GRANT REFERENCES, SELECT, INSERT, DELETE, UPDATE ON swpro.sw_db_bgqueue_3 TO USER
swpro, USER swuser/
```

- 
3. Use the *SWDIR\util\swadm* utility to add a new message queue called BGMBBOX3, which uses the *sw\_db\_bgqueue\_3* queue table.
- 

```
cd SWDIR\util
swadm ADD_QUEUE BGMBBOX3 Local 0003:swpro.sw_db_bgqueue_3
```

- 
4. Add the BGMBBOX3 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).



## Chapter 4      **Sequence Numbers**

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

### Topics

---

- [About Sequence Numbers, page 44](#)
- [Table Relationships, page 46](#)
- [sequences, page 47](#)

# 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 <sup>1</sup>        | JMS provider                       |
| destination_id  | eaiws_jms_destinati<br>on <sup>1</sup> | JMS endpoints for for JMS provider |

1. Only created if TIBCO iProcess Technology Plugins are installed.

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.



For more information about these stored procedures please see the database creation script (`init2Kdb2.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 database-enforced relationships, that is, foreign keys, 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:<br>1 (o_reqid)<br>2 (casenum)<br>3 (proc_id)<br>4 (wait_id)<br>5 (def_id)<br>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:<br>REQID<br>CNUM<br>PROC<br>WAIT<br>CDQP_DEF<br>CDQP<br><b>Note:</b> This value is not currently used by the iProcess Suite. |

**Primary Key** None.

**Foreign Keys** None.

**Indexes** The following clustered index is defined for this table.

| Index Name   | Column(s) Indexed |
|--------------|-------------------|
| idx_sequence | 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.  |

## Chapter 5      **Procedures**

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

### Topics

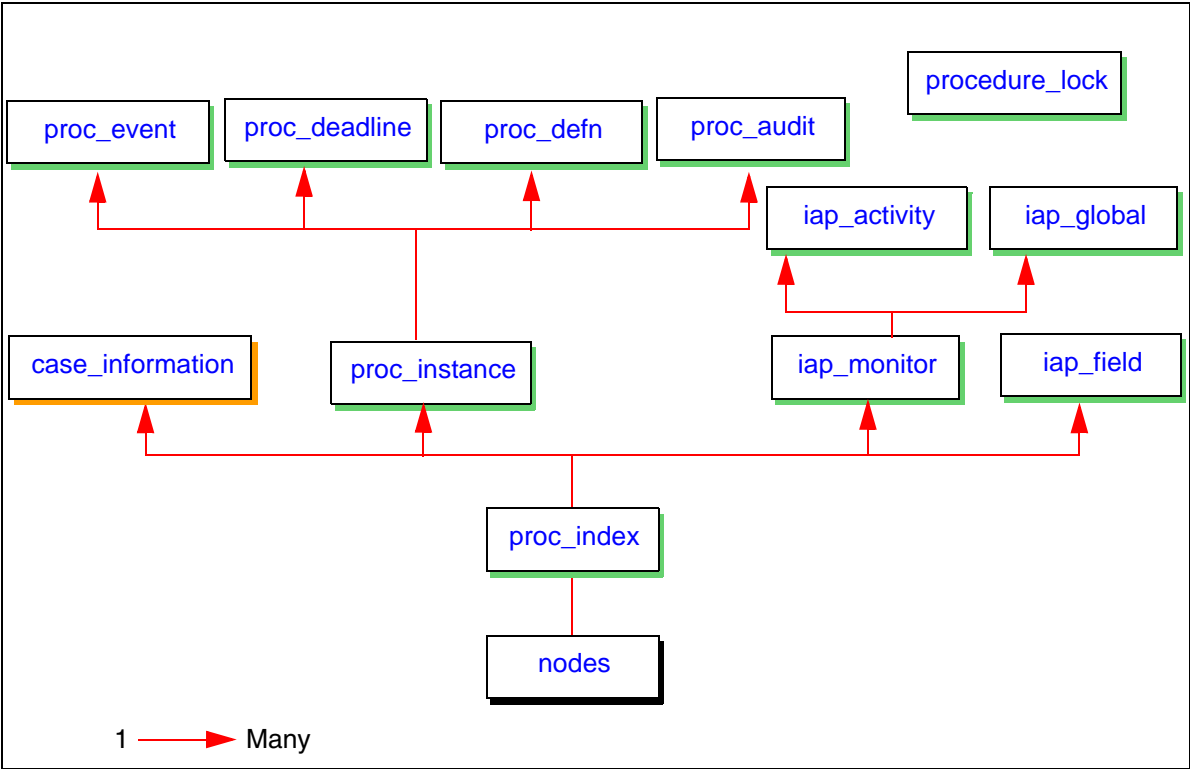
---

- [Table Relationships, page 50](#)
- [proc\\_index, page 51](#)
- [iap\\_monitor, page 55](#)
- [iap\\_field, page 57](#)
- [iap\\_activity, page 59](#)
- [iap\\_global, page 61](#)
- [proc\\_version, page 63](#)
- [procedure\\_lock, page 66](#)
- [proc\\_instance, page 68](#)
- [proc\\_audit, page 70](#)
- [proc\\_defn, page 72](#)
- [proc\\_deadline, page 75](#)
- [proc\\_event, page 77](#)
- [wqd\\_delta\\_subscriptions, page 80](#)

# Table Relationships

The following diagram shows how the tables described in this chapter 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.



## proc\_index

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



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)      ,
  proc_desc        VARCHAR(24)     ,
  proc_owner       VARCHAR(49)     ,
  dir_name         VARCHAR(12)     ,
  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)    )
```

| Column                       | Description   |
|------------------------------|---|
| <code>node_id</code>         | ID of the node that this procedure is defined on, as defined in the <a href="#">nodes</a> table.            |
| <code>proc_id</code>         | Unique ID of this procedure, generated from the <a href="#">sequences</a> table.                            |
| <code>proc_used_count</code> | Not used.   |
| <code>proc_name</code>       | Name of this procedure.<br><b>Note:</b> Internal procedures are prefixed with a dollar sign (\$) character. |
| <code>proc_desc</code>       | Description of this procedure.  |
| <code>proc_owner</code>      | Name of the owner of this procedure, as defined in the <a href="#">user_names</a> table.                    |
| <code>dir_name</code>        | Not used.   |

| Column         | Description   |
|----------------|---|
| 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"> <li>0 means that the field is treated as an error and the step is delivered to the undelivered queue.</li> <li>1 means that the field is not treated as an error.</li> </ul> |
| 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.   |
| delay_purge    | Flag that defines whether or not to delay the auto-purge operation. <ul style="list-style-type: none"> <li>0 means that the auto-purge operation is not delayed.</li> <li>1 means that the auto-purge operation is delayed.</li> </ul>  |

| Column      | Description  |
|-------------|--|
| delay_value | <p>The value of the delay_value column is:</p> <ul style="list-style-type: none"> <li>If the value of the delay_purge column is 0, then the value of the delay_value column is NULL.</li> <li>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"> <li>The period of the delay in days.</li> <li>Delayed date and time expressions:<br/><i>date expression^time expression</i></li> </ul> </li> </ul> |

**Primary Key** The following primary key is defined for this table.

| Key Name      | Column(s)          |
|---------------|--------------------|
| pk_proc_index | proc_id<br>node_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                   | Column(s) | Referenced in Table... |
|----------------------------|-----------|------------------------|
| fk_proc_index <sup>1</sup> | node_id   | nodes                  |

1. This key enforces the DELETE CASCADE referential action.

**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. |

| A row is... | When... |
|-------------|---------|
| 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  |
|-------------------------|--|
| <code>node_id</code>    | ID of the node that this procedure is defined on, as defined in the <a href="#">nodes</a> table. |
| <code>proc_id</code>    | Unique ID of this procedure, generated from the <a href="#">sequences</a> table.                 |
| <code>monitor_id</code> | 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)               |
|-----------------------------|-------------------------|
| <code>pk_iap_monitor</code> | <code>monitor_id</code> |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                                 | Column(s)                                    | Referenced in Table...                              |
|--|--|---|
| <code>fk_iap_monitor</code> <sup>1</sup> | <code>proc_id</code><br><code>node_id</code> | <a href="#">proc_index</a><br><a href="#">nodes</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name                      | Column(s) Indexed                            |
|---------------------------------|--|
| <code>idx_iap_monitor_fk</code> | <code>proc_id</code><br><code>node_id</code> |

**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 <a href="#">iap_monitor</a> 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><br><code>field_name</code> |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                               | Column(s)               | Referenced in Table...      |
|--|-------------------------|-----------------------------|
| <code>fk_iap_field</code> <sup>1</sup> | <code>monitor_id</code> | <a href="#">iap_monitor</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name                    | Column(s) Indexed       |
|-------------------------------|-------------------------|
| <code>idx_iap_field_fk</code> | <code>monitor_id</code> |

**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. |

| A row is... | When...                        |
|-------------|--------------------------------|
| 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 <a href="#">iap_monitor</a> 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)   |
|------------------------------|---|
| <code>pk_iap_activity</code> | <code>monitor_id</code><br><code>activity_id</code><br><code>step_name</code> |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                                  | Column(s)               | Referenced in Table...      |
|---|-------------------------|-----------------------------|
| <code>fk_iap_activity</code> <sup>1</sup> | <code>monitor_id</code> | <a href="#">iap_monitor</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name                       | Column(s) Indexed                            |
|----------------------------------|--|
| <code>idx_iap_activity_fk</code> | <code>proc_id</code><br><code>node_id</code> |

**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   |
|-------------------------|---|
| <code>node_id</code>    | ID of the node that this procedure is stored on, as defined in the <a href="#">nodes</a> table. |
| <code>proc_id</code>    | Unique ID of this procedure, generated from the <a href="#">sequences</a> 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_global</code> | <code>proc_id</code><br><code>node_id</code><br><code>field_name</code> |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                                | Column(s)                                    | Referenced in Table...                              |
|---|--|---|
| <code>fk_iap_global</code> <sup>1</sup> | <code>proc_id</code><br><code>node_id</code> | <a href="#">proc_index</a><br><a href="#">nodes</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name                     | Column(s) Indexed                            |
|--------------------------------|--|
| <code>idx_iap_global_fk</code> | <code>proc_id</code><br><code>node_id</code> |

**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).



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          ,
  pv_status        SMALLINT         ,
  pv_user          VARCHAR(49)      ,
  pv_comment       VARCHAR(128)     NULL,
  pv_created       TIMESTAMP        ,
  pv_modified      TIMESTAMP        ,
  pv_released      TIMESTAMP        ,
  pv_withdrawn     TIMESTAMP        ,
  pv_is_subproc    SMALLINT         )
```

| Column                  | Description  |
|-------------------------|--|
| <code>node_id</code>    | ID of the node that this procedure is stored on, as defined in the <a href="#">nodes</a> table.                                      |
| <code>proc_id</code>    | Procedure number of the procedure associated with this version, as defined in the <a href="#">proc_index</a> table.                  |
| <code>major_vers</code> | Major version number of this version.  |
| <code>minor_vers</code> | Minor version number of this version.  |
| <code>pd_version</code> | Instance number of the procedure definition that corresponds to this version, as defined in the <a href="#">proc_instance</a> table. |
| <code>pv_status</code>  | Status of this version. Either: Released (0), Incomplete (1), Unreleased (2), Model (3) or Withdrawn (14).                           |
| <code>pv_user</code>    | Name of the user who created this version, as defined in the <a href="#">user_names</a> table.                                       |
| <code>pv_comment</code> | Comment describing this version.   |

| Column        | Description   |
|---------------|---|
| 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<br>proc_id<br>major_vers<br>minor_vers |

**Foreign Keys**      The following foreign key is defined for this table.

| Key Name                     | Column(s)          | Referenced in Table...     |
|------------------------------|--------------------|----------------------------|
| fk_proc_version <sup>1</sup> | node_id<br>proc_id | <a href="#">proc_index</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes**      The following index is defined for this table.

| Index Name       | Column(s) Indexed  |
|------------------|--------------------|
| idx_proc_vers_fk | node_id<br>proc_id |

**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. |

| A row is... | When...                            |
|-------------|------------------------------------|
| 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)      ,
    lock_date        TIMESTAMP        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 <a href="#">nodes</a> table.  |
| proc_id     | ID of this procedure, as defined in the <a href="#">proc_index</a> 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 <a href="#">user_names</a> 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"><li>• 0 not locked.</li><li>• 1 locked by the TIBCO iProcess Modeler.</li><li>• 2 locked by SWDIR\bin\swutil IMPORT.</li></ul> |

**Primary Key** The following primary key is defined for this table.

| Key Name          | Column(s)          |
|-------------------|--------------------|
| pk_procedure_lock | proc_id<br>node_id |

**Foreign Keys** 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).



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)      ,
  pi_rpa_start     SMALLINT         ,
  pi_rpa_admin     SMALLINT         ,
  pi_has_eis_objs  SMALLINT         ,
  pi_has_subprocs  SMALLINT         )
```

| Column        | Description   |
|---------------|---|
| node_id       | ID of the node that this instance is stored on, as defined in the <a href="#">nodes</a> table.  |
| proc_id       | Procedure number of the procedure associated with this instance, as defined in the <a href="#">proc_index</a> table.                    |
| major_vers    | Major version number of the version associated with this instance, as defined in the <a href="#">proc_version</a> table.                |
| minor_vers    | Minor version number of the version associated with this instance, as defined in the <a href="#">proc_version</a> 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. |

| Column          | Description  |
|-----------------|--|
| 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<br>proc_id<br>pd_version |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                      | Column(s)          | Referenced in Table... |
|-------------------------------|--------------------|------------------------|
| fk_proc_instance <sup>1</sup> | node_id<br>proc_id | proc_index             |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name       | Column(s) Indexed  |
|------------------|--------------------|
| idx_proc_inst_fk | node_id<br>proc_id |

**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           ,  
    pa_comment         VARCHAR(128)      NULL,  
    pa_event           SMALLINT           ,  
    pa_date            TIMESTAMP         ,  
    pa_user            VARCHAR(24)       )
```

| Column     | Description  |
|------------|--|
| node_id    | ID of the node that this audit event is stored on, as defined in the <a href="#">nodes</a> table.                                    |
| proc_id    | Procedure number of the procedure associated with this audit event, as defined in the <a href="#">proc_index</a> table.              |
| major_vers | Major version number of the version associated with this audit event, as defined in the <a href="#">proc_version</a> table.          |
| minor_vers | Minor version number of the version associated with this audit event, as defined in the <a href="#">proc_version</a> table.          |
| pd_version | Instance number of the procedure definition associated with this audit event, as defined in the <a href="#">proc_instance</a> 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.   |

| Column  | Description   |
|---------|---|
| pa_user | Name of the user who performed the audit event, as defined in the <a href="#">user_names</a> table. |

**Primary Key** None.

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                   | Column(s)                        | Referenced in Table...        |
|----------------------------|----------------------------------|-------------------------------|
| fk_proc_audit <sup>1</sup> | node_id<br>proc_id<br>pd_version | <a href="#">proc_instance</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name        | Column(s) Indexed                |
|-------------------|----------------------------------|
| idx_proc_audit_fk | node_id<br>proc_id<br>pd_version |

**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             BLOB(30K)         NOT NULL NOT LOGGED)
```

| Column     | Description   |
|------------|---|
| node_id    | ID of the node that this procedure definition is defined on, as defined in the <a href="#">nodes</a> table.   |
| proc_id    | ID of the procedure that this procedure definition relates to, as defined in the <a href="#">proc_index</a> table.  |
| pd_version | ID of the procedure instance that this row relates to, as defined in the <a href="#">proc_instance</a> table.   |
| pd_type    | Type of procedure definition data stored in this row. Either: <ul style="list-style-type: none"><li>0 <i>pro</i> data (textual procedure definition)</li><li>1 <i>lst</i> data (binary procedure definition)</li><li>2 <i>nod</i> data (not used)</li><li>3 <i>gwd</i> data (iProcess Modeler layout information)</li><li>4 <i>nod</i> data (not used)</li><li>5 VBA project data (VBA project files)</li></ul> |
| pd_index   | Index number into the set of rows that make up this procedure definition. <ul style="list-style-type: none"><li>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 pd_index value.</li></ul>  |

| Column       | Description  |
|--------------|--|
| 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 | If pd_type is 5, contains the sub-directory path (relative to SWDIR\projects) where any VBA project files related to this procedure definition are stored. Filenames are stored in pd_file.<br>If pd_type is 0 to 4, this field contains a hyphen. |
| pd_file      | If pd_type 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 pd_directory.<br>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<br>pd_index<br>pd_type<br>node_id<br>pd_version<br>pd_directory<br>pd_file |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                  | Column(s)                        | Referenced in Table... |
|---------------------------|----------------------------------|------------------------|
| fk_proc_defn <sup>1</sup> | node_id<br>proc_id<br>pd_version | proc_instance          |

1. This key enforces the DELETE CASCADE referential action.

**Indexes**      The following index is defined for this table.

| Index Name       | Column(s) Indexed                           |
|------------------|---|
| idx_proc_defn    | proc_id<br>pd_version<br>pd_type<br>node_id |
| idx_proc_defn_fk | proc_id<br>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 <a href="#">nodes</a> table.   |
| <code>proc_id</code>    | Unique ID of this procedure, generated from the <a href="#">sequences</a> table.   |
| <code>major_vers</code> | The major version number of the procedure that this case belongs to, as defined in the <a href="#">proc_version</a> table.   |
| <code>minor_vers</code> | The minor version number of the procedure that this case belongs to, as defined in the <a href="#">proc_version</a> table.   |
| <code>pd_version</code> | Instance number of the procedure definition, as defined in the <a href="#">proc_instance</a> table.  |
| <code>dead_name</code>  | The name of the case deadline.   |
| <code>event_name</code> | The name of the event step that is triggered when the case deadline expires.   |
| <code>dead_value</code> | <p>The value of the case deadline. The value is specified in one of the following formats:</p> <ul style="list-style-type: none"> <li>If the case deadline is specified as a period, then the value is in the format:<br/> <i>minutes^hours^days^weeks^months^years</i></li> <li>If the case deadline is specified as an expression, then the value is in the format:<br/> <i>date expression^time expression</i></li> </ul> |

**Primary Key**      The following primary key is defined for this table.

| Key Name         | Column(s)                                     |
|------------------|---|
| pk_proc_deadline | node_id<br>proc_id<br>pd_version<br>dead_name |

**Foreign Keys**      The following foreign key is defined for this table.

| Key Name                      | Column(s)                        | Referenced in Table... |
|-------------------------------|----------------------------------|------------------------|
| fk_proc_deadline <sup>1</sup> | node_id<br>proc_id<br>pd_version | proc_instance          |

1. This key enforces the DELETE CASCADE referential action.

**Indexes**      The following index is defined for this table.

| Key Name       | Column(s) Indexed               |
|----------------|---------------------------------|
| idx_proc_dl_fk | node_id<br>pro_id<br>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(32)     NOT NULL,
  user_event_name  VARCHAR(32)     NOT NULL)
```

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

| Column          | Description   |
|-----------------|---|
| 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<br>proc_id<br>pd_version<br>eventname |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                   | Column(s)                        | Referenced in Table...        |
|----------------------------|----------------------------------|-------------------------------|
| fk_proc_event <sup>1</sup> | node_id<br>proc_id<br>pd_version | <a href="#">proc_instance</a> |

1. This key enforces the DELETE CASCADE referential action.

**Index** The following index is defined for this table.

| Index Name        | Column(s) Indexed                |
|-------------------|----------------------------------|
| idx_proc_event_fk | node_id<br>proc_id<br>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"> <li>• A new procedure event is added.</li> <li>• A procedure event is modified.</li> </ul> <p><b>Note:</b> 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"> <li>• A new version of a procedure is released.</li> </ul> |
| updated     | Never.   |
| deleted     | <p>If one of the following conditions occurred:</p> <ul style="list-style-type: none"> <li>• A procedure event is deleted.</li> <li>• A procedure event is modified.</li> </ul> <p><b>Note:</b> 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"> <li>• The version of this procedure is deleted.</li> </ul>   |

# 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,  
    wqdid                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     |

**Foreign Keys** None.

**Indexes** None.

## Chapter 6 Procedure Management

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

### Topics

---

- [About Procedure Objects, page 82](#)
- [Table Relationships, page 83](#)
- [pm\\_objects, page 84](#)
- [pm\\_objects\\_lock, page 87](#)
- [pmobjects\\_security, page 90](#)
- [proc\\_mgt\\_hierarchy, page 92](#)

## 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.



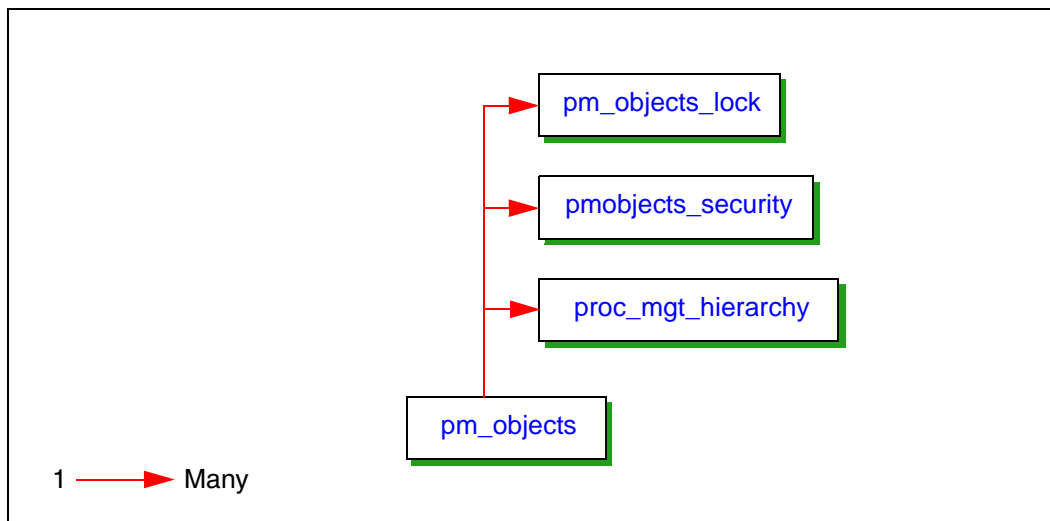
Shortcuts are *not* real procedure objects. They are simply placeholders that allow you to access a procedure object from different locations in the Procedure Management library. Data on shortcuts is only stored in the [proc\\_mgt\\_hierarchy](#) table.

Information about procedure versions is stored in other tables - see [Procedures on page 49](#) for more information.

## Table Relationships

The following diagram shows how the tables described in this chapter 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.



# 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              ,
  version_minor        INTEGER              ,
  icon_mod_time        TIMESTAMP            NOT NULL,
  icon_binary           BLOB(30K)            NOT LOGGED,
  icon_size            INTEGER              NOT NULL,
  object_url           VARCHAR(1000)        ,
  author               VARCHAR(64)         ,
  object_create_time   TIMESTAMP            NOT NULL,
  object_mod_time      TIMESTAMP            NOT NULL,
  release_id           VARCHAR(64)         ,
  security_all         SMALLINT             NOT NULL,
  proc_id              INTEGER              NOT NULL,
  proc_status          SMALLINT             )
```

| Column        | Description  |
|---------------|--|
| object_guid   | Globally unique, system-generated identifier for this procedure object.<br>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.<br><b>Note:</b> 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 <a href="#">proc_version</a> table.<br>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 <a href="#">proc_version</a> table.<br>This value is always 0 if the object is a library. |

| Column                          | Description   |
|---------------------------------|---|
| <code>icon_mod_time</code>      | Time that the icon associated with this procedure object was last modified.   |
| <code>icon_binary</code>        | Binary form of the icon associated with this procedure object.  |
| <code>icon_size</code>          | Size (in bytes) of the icon associated with this procedure object.  |
| <code>object_url</code>         | Usage URL associated with this procedure object.  |
| <code>author</code>             | Value of the Author extended property for this procedure object.  |
| <code>object_create_time</code> | Value of the Date Created extended property for this procedure object, showing the time that this object was created.   |
| <code>object_mod_time</code>    | Value of the Date Modified extended property for this procedure object, showing the time that this procedure object was last modified.  |
| <code>release_id</code>         | Value of the Release Identification extended property for this procedure object.  |
| <code>security_all</code>       | Flag that defines whether (1) or not (0) the OEM lock is set for this procedure object.   |
| <code>proc_id</code>            | Procedure number of the procedure associated with this procedure object, as defined in the <a href="#">proc_index</a> table.<br>This value is always -1 if the object is a library. |
| <code>proc_status</code>        | For internal use only.  |

**Primary Key** The following primary key is defined for this table.

| Key Name                   | Column(s)                |
|----------------------------|--------------------------|
| <code>pk_pm_objects</code> | <code>object_guid</code> |

**Foreign Keys** None.

**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      INTEGER          ,
  lck_owner      VARCHAR(24)      ,
  lck_time       TIMESTAMP        )
```

| Column                   | Description  |
|--------------------------|--|
| <code>object_guid</code> | ID for this procedure object, as defined in the <a href="#">pm_objects</a> table.                          |
| <code>lck_state</code>   | Flag that defines (1) that this procedure object is locked.  |
| <code>lck_owner</code>   | Name of the user who has this procedure object locked, as defined in the <a href="#">user_names</a> table. |
| <code>lck_time</code>    | 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)                |
|---------------------------------|--------------------------|
| <code>pk_pm_objects_lock</code> | <code>object_guid</code> |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                                     | Column(s)                | Referenced in Table...     |
|--|--------------------------|----------------------------|
| <code>fk_pm_objects_lock</code> <sup>1</sup> | <code>object_guid</code> | <a href="#">pm_objects</a> |

1. This key enforces the DELETE CASCADE referential action.

**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 DB2 as the background user.
- 2. 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 may need to refresh the display first.)



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.

# pobjects\_security

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

**Structure** The pobjects\_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 <a href="#">pm_objects</a> 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_sec | object_guid<br>security_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                     | Column(s)   | Referenced in Table...     |
|------------------------------|-------------|----------------------------|
| fk_pobjects_sec <sup>1</sup> | object_guid | <a href="#">pm_objects</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name        | Column(s) Indexed |
|-------------------|-------------------|
| idx_pmobjs_sec_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  |
|-------------|--|
| parent_guid | ID for the parent library, as defined in the <a href="#">pm_objects</a> table.   |
| object_guid | ID for this procedure object, as defined in the <a href="#">pm_objects</a> table.<br><b>Note:</b> If <code>is_shortcut</code> is 1, this value is the identifier of the procedure object pointed to by the shortcut. |
| is_shortcut | 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)                  |
|-----------------------|----------------------------|
| pk_proc_mgt_hierarchy | parent_guid<br>object_guid |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                           | Column(s)   | Referenced in Table...     |
|------------------------------------|-------------|----------------------------|
| fk_proc_mgt_hierarchy <sup>1</sup> | object_guid | <a href="#">pm_objects</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name        | Column(s) Indexed |
|-------------------|-------------------|
| idx_proc_mgt_h_fk | object_guid       |

**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.         |



## Chapter 7      **Cases**

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

### Topics

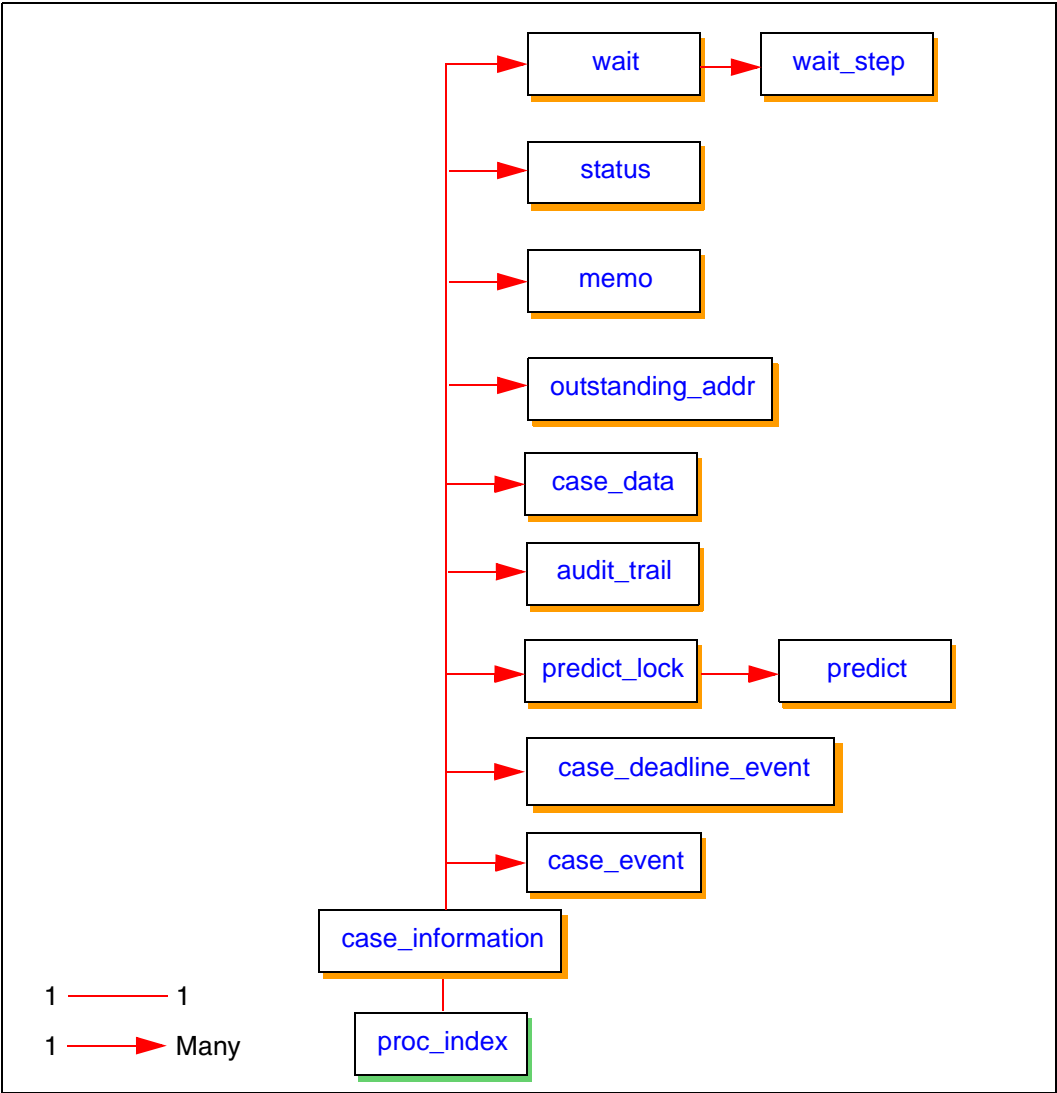
---

- [Table Relationships, page 96](#)
- [case\\_information, page 97](#)
- [outstanding\\_addr, page 100](#)
- [wait, page 103](#)
- [wait\\_step, page 105](#)
- [status, page 107](#)
- [case\\_data, page 109](#)
- [audit\\_trail, page 111](#)
- [memo, page 114](#)
- [predict, page 116](#)
- [predict\\_lock, page 120](#)
- [case\\_deadline\\_event, page 122](#)
- [case\\_event, page 125](#)
- [casenum\\_gaps, page 128](#)

# Table Relationships

The following diagram shows how the tables described in this chapter 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.



## 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    TIMESTAMP        ,
  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          TIMESTAMP        NOT NULL,
  started_usecs    NUMERIC(10)      NOT NULL)
```

| Column                 | Description  |
|------------------------|--|
| <code>node_id</code>   | ID of the node that this case is hosted on, as defined in the <a href="#">nodes</a> table.   |
| <code>proc_id</code>   | ID of the procedure that this case belongs to, as defined in the <a href="#">proc_index</a> table.   |
| <code>casenum</code>   | Unique case number for this case, generated from the <a href="#">sequences</a> table.  |
| <code>starter</code>   | Name of the user who started this case, as defined in the <a href="#">user_names</a> table.  |
| <code>casedesc</code>  | Case description supplied when the case was started.   |
| <code>procflags</code> | <p>The procedure flags that were set at the time the case was started. For internal use only.</p> <p><b>Note:</b> 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.<br>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 <a href="#">proc_version</a> table.  |
| minor_vers      | Minor version number of the version of the procedure that this case belongs to, as defined in the <a href="#">proc_version</a> table.  |
| proc_precedence | Stores the procedure precedence settings for opening sub-cases. One of: <ul style="list-style-type: none"> <li>• 32 - Released only.</li> <li>• 64 - Unreleased &gt; Released.</li> <li>• 96 - Model &gt; Released.</li> <li>• 128 - Unreleased &gt; Model &gt; Released.</li> <li>• 160 - Model &gt; Unreleased &gt; Released.</li> </ul> |
| started         | Date and time that the case was started, to the resolution of a second.<br><b>Note:</b> The started_usec column can be combined with this column to provide resolution to a microsecond.   |
| started_usec    | Number of microseconds since the start of the <i>seconds</i> value specified in the started column.  |

**Primary Key** The following primary key is defined for this table.

| Key Name     | Column(s)                     |
|--------------|-------------------------------|
| pk_case_info | casenum<br>proc_id<br>node_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                         | Column(s)          | Referenced in Table... |
|----------------------------------|--------------------|------------------------|
| fk_case_information <sup>1</sup> | proc_id<br>node_id | proc_index             |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following indexes are defined for this table.

| Index Name       | Column(s) Indexed  |
|------------------|--------------------|
| idx_case_info_fk | proc_id<br>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"> <li>a case or sub-case is closed.</li> <li>a deadline on a case or sub-case is set or expires.</li> <li>a case or sub-case is suspended or re-opened.</li> <li>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.</li> </ul> |
| 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                NUMERIC(15)                NOT NULL GENERATED
  ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1, CACHE 50),
  node_id              INTEGER                     NOT NULL,
  proc_id              INTEGER                     NOT NULL,
  casenum              NUMERIC(20)                 NOT NULL,
  sentdate             TIMESTAMP                   NOT NULL,
  deadline             SMALLINT                    NOT NULL,
  deadline_expired     SMALLINT                    NOT NULL,
  sub_procedure        SMALLINT                    NOT NULL,
  deaddate             TIMESTAMP                   NOT NULL,
  stepname             VARCHAR(8)                  NOT NULL,
  user_name            VARCHAR(64)                  NOT NULL,
  reqid               NUMERIC(20)                  NOT NULL,
  item_suspended      SMALLINT                     ,
  item_withdrawn       SMALLINT                     ,
  array_idx            INTEGER                     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 <a href="#">nodes</a> table.              |
| proc_id          | ID of the procedure that this outstanding step belongs to, as defined in the <a href="#">proc_index</a> table.      |
| casenum          | Number of the case that this outstanding step belongs to, as defined in the <a href="#">case_information</a> 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.                             |

| Column         | Description  |
|----------------|--|
| deaddate       | Date and time that the deadline (if defined) expires on this outstanding step.<br>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 <a href="#">user_names</a> table.   |
| reqid          | Unique ID for this work item, generated from the <a href="#">sequences</a> table.  |
| item_suspended | Flag that defines whether (1) or not (0) this outstanding step is currently suspended.<br><b>Note:</b> <code>item_suspended</code> 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"> <li>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.</li> <li>-1, otherwise.</li> </ul> |

**Primary Key** None.

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                      | Column(s)                     | Referenced in Table...           |
|-------------------------------|-------------------------------|----------------------------------|
| fk_outstand_addr <sup>1</sup> | casenum<br>proc_id<br>node_id | <a href="#">case_information</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes**      The following indexes are defined for this table.

| Index Name                     | Column(s) Indexed             |
|--------------------------------|-------------------------------|
| idx_deadline_date              | deaddate                      |
| idx_outstand_addr <sup>1</sup> | rowid                         |
| idx_outstd_addr_fk             | casenum<br>proc_id<br>node_id |

1. UNIQUE index.

**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><b>Note:</b></p> <ul style="list-style-type: none"><li>• if a step has multiple addressees one row is added per addressee.</li><li>• for a dynamic sub-procedure, one row is added per called sub-procedure.</li><li>• for a graft step, one row is added per grafted sub-procedure or external step.</li></ul> |
| updated     | <p>any of the following occur:</p> <ul style="list-style-type: none"><li>• a deadline on an outstanding step expires.</li><li>• a case is suspended or re-opened.</li><li>• an outstanding step is withdrawn.</li></ul>   |
| 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)           ,
  expression       VARCHAR(200)         ,
  type             SMALLINT             NOT NULL)
```

| Column     | Description   |
|------------|---|
| wait_id    | Unique ID for this wait, generated from the <a href="#">sequences</a> table.  |
| node_id    | ID of the node that this wait is hosted on, as defined in the <a href="#">nodes</a> table.  |
| proc_id    | ID of the procedure that this wait belongs to, as defined in the <a href="#">proc_index</a> table.  |
| casenum    | Case number that this wait belongs to, as defined in the <a href="#">case_information</a> 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<br>node_id |

**Foreign Keys**      The following foreign key is defined for this table.

| Key Name             | Column(s)          | Referenced in Table...           |
|----------------------|--------------------|----------------------------------|
| fk_wait <sup>1</sup> | node_id<br>proc_id | <a href="#">case_information</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes**      The following indexes are defined for this table.

| Index Name  | Column(s) Indexed             |
|-------------|-------------------------------|
| idx_wait_fk | casenum<br>proc_id<br>node_id |
| idx_wait    | casenum<br>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  |
|----------------------|--|
| <code>node_id</code> | ID of the node that the wait is hosted on, as defined in the <a href="#">nodes</a> table.  |
| <code>proc_id</code> | ID of the procedure that the wait belongs to, as defined in the <a href="#">proc_index</a> table.                                  |
| <code>wait_id</code> | ID of the wait, as defined in the <a href="#">wait_step</a> table.   |
| <code>step_id</code> | Number of the step that is being waited for, as defined (by the <code>step_num</code> column) in the <a href="#">status</a> table. |

**Primary Key** The following primary key is defined for this table.

| Key Name                  | Column(s)                                    |
|---------------------------|--|
| <code>pk_wait_step</code> | <code>wait_id</code><br><code>step_id</code> |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                               | Column(s)                                    | Referenced in Table... |
|--|--|------------------------|
| <code>fk_wait_step</code> <sup>1</sup> | <code>node_id</code><br><code>wait_id</code> | <a href="#">wait</a>   |

1. This key enforces the `DELETE CASCADE` referential action.

**Indexes**      The following indexes are defined for this table.

| Index Name       | Column(s) Indexed  |
|------------------|--------------------|
| idx_wait_step_fk | wait_id<br>node_id |
| idx_wait_step    | wait_id<br>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  |
|--------------------------|--|
| <code>node_id</code>     | ID of the node that this step is hosted on, as defined in the <a href="#">nodes</a> table.                         |
| <code>proc_id</code>     | ID of the procedure that this step belongs to, as defined in the <a href="#">proc_index</a> table.                 |
| <code>casenum</code>     | Case number that this step belongs to, as defined in the <a href="#">case_information</a> table.                   |
| <code>step_num</code>    | Place number for this step (a unique ID that does not change between edits of a procedure). For internal use only. |
| <code>step_status</code> | 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)   |
|------------------------|---|
| <code>pk_status</code> | <code>casenum</code><br><code>proc_id</code><br><code>step_num</code><br><code>node_id</code> |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                            | Column(s)  | Referenced in Table...           |
|-------------------------------------|--|----------------------------------|
| <code>fk_status</code> <sup>1</sup> | <code>node_id</code><br><code>proc_id</code><br><code>casenum</code> | <a href="#">case_information</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes**      The following index is defined for this table.

| Index Name    | Column(s) Indexed             |
|---------------|-------------------------------|
| idx_status_fk | casenum<br>proc_id<br>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)     ,
  field_value_N    VARCHAR(255)     )
```

| Column                     | Description   |
|----------------------------|---|
| <code>node_id</code>       | ID of the node that this field is hosted on, as defined in the <a href="#">nodes</a> table.   |
| <code>proc_id</code>       | ID of the procedure that this field belongs to, as defined in the <a href="#">proc_index</a> table.   |
| <code>casenum</code>       | Case number that this field belongs to, as defined in the <a href="#">case_information</a> table.   |
| <code>field_name</code>    | Name of this field.   |
| <code>field_value</code>   | Value of this field.  |
| <code>field_value_N</code> | <p>“Normalized” value of the <code>field_value</code> value. That is:</p> <ul style="list-style-type: none"> <li>• Date values are stored as YYYY-MM-DD.</li> <li>• Numeric values are stored as padded strings.</li> <li>• Time and String values are not changed.</li> </ul> <p><b>Note:</b> 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<br>proc_id<br>node_id<br>field_name |

**Foreign Keys**      The following foreign key is defined for this table.

| Key Name                  | Column(s)                     | Referenced in Table...           |
|---------------------------|-------------------------------|----------------------------------|
| fk_case_data <sup>1</sup> | node_id<br>proc_id<br>casenum | <a href="#">case_information</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes**      The following indexes are defined for this table.

| Index Name                    | Column(s) Indexed                                 |
|-------------------------------|---|
| idx_case_data_fk              | casenum<br>proc_id<br>node_id                     |
| idx_cd_cn_p_fnfv <sup>1</sup> | field_name<br>field_value_N<br>casenum<br>proc_id |

1. This 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.

**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. |

## 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       TIMESTAMP        NOT NULL,
  stepdesc         VARCHAR(24)      ,
  user_name        VARCHAR(64)      ,
  stepname         VARCHAR(8)       ,
  audit_usecs      NUMERIC(10)      NOT NULL,
  major_vers       INTEGER          NOT NULL,
  minor_vers       INTEGER          NOT NULL)
```

| Column                  | Description  |
|-------------------------|--|
| <code>node_id</code>    | ID of the node that this audit event is hosted on, as defined in the <a href="#">nodes</a> table.  |
| <code>proc_id</code>    | ID of the procedure that this audit event belongs to, as defined in the <a href="#">proc_index</a> table.  |
| <code>casenum</code>    | Case number that this audit event belongs to, as defined in the <a href="#">case_information</a> table.  |
| <code>type_id</code>    | <p>ID of the audit event that occurred. Either:</p> <ul style="list-style-type: none"> <li>a system-defined audit event (<math>\leq 255</math>), as defined in the <code>SWDIR\etc\language.lng\audit.mes</code> file.</li> <li>a custom, application-defined event (256-999), as defined in the <code>SWDIR\etc\language.lng\auditusr.mes</code> file.</li> </ul> <p><b>Note:</b> See "Defining Audit Trail Entries" in <i>TIBCO iProcess swutil and swbatch Reference Guide</i> for more information about system-defined and application-defined audit trail entries.</p> |
| <code>audit_date</code> | <p>Date and time that this audit event occurred.</p> <p><b>Note:</b> The <code>audit_usecs</code> column can be combined with this column to provide resolution to a microsecond.</p>  |

| Column      | Description  |
|-------------|--|
| stepdesc    | If type_id is: <ul style="list-style-type: none"><li>• &lt;= 255, the step description of the step that this audit event occurred to.</li><li>• =&gt; 256, a user-defined string, containing for example the description of this audit event.</li></ul>  |
| user_name   | If type_id is: <ul style="list-style-type: none"><li>• &lt;= 255, the name of the user who performed this audit event, as defined in the <a href="#">user_names</a> table.</li><li>• =&gt; 256, a user-defined string, containing for example the name of the user who performed this audit event.</li></ul> |
| stepname    | Name of the step that this audit event occurred for.<br>For internal use only.   |
| audit_usecs | Number of microseconds since the start of the <i>seconds</i> value specified in the audit_date column.   |
| major_vers  | Major version number of the version of the procedure that this audit event belongs to, as defined in the <a href="#">proc_version</a> table.   |
| minor_vers  | Minor version number of the version of the procedure that this audit event belongs to, as defined in the <a href="#">proc_version</a> table.   |

**Primary Key**      None.

**Foreign Keys**      The following foreign key is defined for this table.

| Key Name                    | Column(s)                     | Referenced in Table...           |
|-----------------------------|-------------------------------|----------------------------------|
| fk_audit_trail <sup>1</sup> | node_id<br>proc_id<br>casenum | <a href="#">case_information</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes**      The following index is defined for this table.

| Index Name         | Column(s) Indexed             |
|--------------------|-------------------------------|
| idx_audit_trail_fk | casenum<br>proc_id<br>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.



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        BLOB(30K)       NOT NULL NOT LOGGED,  
  array_idx        INTEGER          NOT NULL)
```

| Column     | Description   |
|------------|---|
| node_id    | ID of the node that this memo is hosted on, as defined in the <a href="#">nodes</a> table.  |
| proc_id    | ID of the procedure that this memo belongs to, as defined in the <a href="#">proc_index</a> table.  |
| casenum    | Case number that this memo belongs to, as defined in the <a href="#">case_information</a> 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. <ul style="list-style-type: none"><li>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.</li></ul> |
| 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"><li>The array element index number of the memo.</li><li>-1, if the memo is not an array memo field.</li></ul>   |

**Primary Key** The following primary key is defined for this table.

| Key Name | Column(s)   |
|----------|---|
| pk_memo  | casenum<br>memo_id<br>memo_index<br>proc_id<br>node_id<br>array_idx |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name             | Column(s)                     | Referenced in Table...           |
|----------------------|-------------------------------|----------------------------------|
| fk_memo <sup>1</sup> | node_id<br>proc_id<br>casenum | <a href="#">case_information</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name  | Column(s) Indexed             |
|-------------|-------------------------------|
| idx_memo_fk | casenum<br>proc_id<br>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"> <li>a memo field is first assigned.</li> <li>a memo field is modified. (All rows for the memo are deleted and then re-added.)</li> </ul> |
| updated     | never.  |
| deleted     | either: <ul style="list-style-type: none"> <li>a memo field is modified. (All rows for the memo are deleted and then re-added.)</li> <li>a case is purged.</li> </ul>               |

# 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          NUMERIC(5)          NOT NULL,
  proc_num         NUMERIC(5)          NOT NULL,
  case_num         NUMERIC(15)         NOT NULL,
  parent_proc_num  NUMERIC(5)          NOT NULL,
  parent_case_num  NUMERIC(15)         NOT NULL,
  main_proc_num    NUMERIC(5)          NOT NULL,
  main_case_num    NUMERIC(15)         NOT NULL,
  step_name        VARCHAR(8)          NOT NULL,
  step_desc        VARCHAR(24)         ,
  step_desc2       VARCHAR(24)         ,
  step_addr        VARCHAR(49)         NOT NULL,
  step_durn_secs   NUMERIC(10)         ,
  step_durn_usecs  NUMERIC(10)         ,
  step_start       TIMESTAMP           NOT NULL,
  step_start_usecs NUMERIC(10)         NOT NULL,
  step_end         TIMESTAMP           NOT NULL,
  step_end_usecs   NUMERIC(10)         NOT NULL,
  field_name       VARCHAR(31)         ,
  field_value      VARCHAR(255)        )
```

| Column          | Description  |
|-----------------|--|
| node_id         | ID of the node that this predicted work item is hosted on, as defined in the <a href="#">nodes</a> table.  |
| proc_num        | ID of the procedure associated with this predicted work item, as defined in the <a href="#">proc_index</a> table.  |
| case_num        | Either: <ul style="list-style-type: none"><li>• Case number of the case associated with this predicted work item, as defined in the <a href="#">case_information</a> table.</li><li>• 0, if this is a predicted work item in a future sub-case, rather than in a currently outstanding sub-case.</li></ul> |
| parent_proc_num | ID of the parent procedure associated with this predicted work item, as defined in the <a href="#">proc_index</a> table, if <code>proc_num</code> is a sub-procedure.  |

| Column           | Description   |
|------------------|---|
| parent_case_num  | Either: <ul style="list-style-type: none"> <li>ID of the parent case associated with this predicted work item, as defined in the <a href="#">case_information</a> table, if case_num is a sub-case.</li> <li>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.</li> </ul> |
| main_proc_num    | ID of the procedure associated with the main case that generated this predicted work item, as defined in the <a href="#">proc_index</a> table.  |
| main_case_num    | ID of the main case that generated this predicted work item, as defined in the <a href="#">case_information</a> 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.  |
| 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.<br><b>Note:</b> 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.<br><b>Note:</b> 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.  |

| Column         | Description  |
|----------------|--|
| 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.<br><br><b>Note:</b> 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 seconds 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.

**Foreign Keys**      The following foreign key is defined for this table.

| Key Name                | Column(s)                                 | Referenced in Table...       | Column(s)                       |
|-------------------------|---|------------------------------|---------------------------------|
| fk_predict <sup>1</sup> | main_case_num<br>main_proc_num<br>node_id | <a href="#">predict_lock</a> | case_num<br>proc_num<br>node_id |

1. This key enforces the DELETE CASCADE referential action.

**Indexes**      The following index is defined for this table.

| Index Name     | Column(s) Indexed                         |
|----------------|---|
| idx_predict_fk | main_case_num<br>main_proc_num<br>node_id |

**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.<br><b>Note:</b> 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.<br><b>Note:</b> 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). |



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          NUMERIC(5)          NOT NULL,  
    proc_num         NUMERIC(5)          NOT NULL,  
    case_num         NUMERIC(15)         NOT NULL)
```

| Column   | Description  |
|----------|--|
| node_id  | ID of the node that this prediction lock is hosted on, as defined in the <a href="#">nodes</a> table.                        |
| proc_num | ID of the procedure that this prediction lock applies to, as defined in the <a href="#">proc_index</a> table.                |
| case_num | Case number of the main case that this prediction lock applies to, as defined in the <a href="#">case_information</a> table. |

**Primary Key** The following primary key is defined for this table.

| Key Name        | Column(s)                       |
|-----------------|---------------------------------|
| pk_predict_lock | node_id<br>proc_num<br>case_num |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                         | Column(s)                       | Referenced in Table...           | Column(s)                     |
|----------------------------------|---------------------------------|----------------------------------|-------------------------------|
| fk_predict_loc<br>k <sup>1</sup> | node_id<br>proc_num<br>case_num | <a href="#">case_information</a> | node_id<br>proc_id<br>casenum |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name       | Column(s) Indexed               |
|------------------|---------------------------------|
| idx_pred_lock_fk | case_num<br>proc_num<br>node_id |

**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.<br><br><b>Note:</b> 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. |
| 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(512)     NOT NULL)
```

| Column     | Description  |
|------------|--|
| node_id    | ID of the node that this procedure is defined on, as defined in the <a href="#">nodes</a> table.   |
| proc_id    | Unique ID of this procedure, generated from the <a href="#">sequences</a> table.   |
| casenum    | The number of the case that this case deadline belongs to, as defined in the <a href="#">case_information</a> table.   |
| dead_id    | For internal use only. This column is referenced from the stepname column in the <a href="#">outstanding_addr</a> table.   |
| dead_name  | The name of the case deadline.   |
| event_name | The name of the event step.  |
| dead_value | <div>The value of the case deadline when the case is running. The value is specified in one of the following formats:</div> <ul style="list-style-type: none"><li>If the case deadline is specified as a period, then the value is in the format:<br/><i>minutes^hours^days^weeks^months^years</i></li><li>If the case deadline is specified as an expression, then the value is in the format:<br/><i>date expression^time expression</i></li></ul> |

**Primary Key** The following primary key is defined for this table.

| Key Name         | Column(s)                                |
|------------------|--|
| pk_case_dl_event | node_id<br>proc_id<br>casenum<br>dead_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                      | Column(s)                     | Referenced in Table...           |
|-------------------------------|-------------------------------|----------------------------------|
| fk_case_dl_event <sup>1</sup> | casenum<br>proc_id<br>node_id | <a href="#">case_information</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name     | Column(s) Indexed             |
|----------------|-------------------------------|
| idx_case_dl_fk | casenum<br>proc_id<br>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"> <li>A case is starting and its deadline is defined in the procedure.</li> <li>The CreateCaseDeadline expression is called in the EAI step.</li> </ul> |
| updated     | The UpdateCaseDeadline expression is called in the EAI step.  |

| A row is... | When...  |
|-------------|--|
| deleted     | <div>If one of the following conditions occurs:<ul style="list-style-type: none"><li>• The DeleteCaseDeadline expression is called in the EAI step.</li><li>• The case deadline expired and an event is triggered.</li><li>• The case is closed.</li></ul></div> |

## 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 <a href="#">nodes</a> table.                                     |
| proc_id    | ID of the procedure that this procedure event belongs to, as defined in the <a href="#">proc_index</a> table.                  |
| major_vers | Major version number of the procedure version that this case belongs to, as defined in the <a href="#">proc_version</a> table. |
| minor_vers | Minor version number of the procedure version that this case belongs to, as defined in the <a href="#">proc_version</a> table. |

| Column          | Description   |
|-----------------|---|
| 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"><li>BeforePurge</li><li>BeforeClose</li><li>AfterClose</li><li>BeforeResurrect</li><li>AfterResurrect</li><li>BeforeSuspend</li><li>AfterSuspend</li><li>BeforeResume</li><li>AfterResume</li></ul>   |
| user_event_name | <p>The name of the event step which you set for the procedure event.</p>  |
| casenum         | <p>ID of the case that this event belongs to, as defined in the <a href="#">case_information</a> table.</p>   |
| 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"><li><b>2</b> the triggered event is in the processing state.</li><li><b>3</b> the triggered event is finished.</li><li><b>4</b> the triggered event is cancelled.</li><li><b>-1</b> the triggered event failed.</li></ul>                                     |
| 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<br>proc_id<br>casenum<br>eventname |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                   | Column(s)                     | Referenced in Table...           |
|----------------------------|-------------------------------|----------------------------------|
| fk_case_event <sup>1</sup> | casenum<br>proc_id<br>node_id | <a href="#">case_information</a> |

1. This key enforces the DELETE CASCADE referential action.

**Index** The following index is defined for this table.

| Index Name        | Column(s) Indexed             |
|-------------------|-------------------------------|
| idx_case_event_fk | casenum<br>proc_id<br>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 cancelled.   |
| deleted     | If one of the following conditions occurred: <ul style="list-style-type: none"> <li>The case is purged.</li> <li>The procedure event is finished.</li> <li>The procedure event failed.</li> <li>The procedure event is cancelled.</li> </ul> |

## 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.



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                       |
|-------------|-----------------------------------|
| casenum_min | The minimum case number in a gap. |
| casenum_max | The maximum case number in a gap. |

**Primary Key** The following primary key is defined for this table.

| Key Name        | Column(s)   |
|-----------------|-------------|
| pk_casenum_gaps | casenum_min |

**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 CASENUM_FIND_GAPS stored procedure. |
| updated     | running TIBCO iProcess Engine.                  |
| deleted     | running TIBCO iProcess Engine.                  |

**See Also** [CASENUM\\_FIND\\_GAPS](#)



## Chapter 8      **Work Items**

This chapter 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”).

### Topics

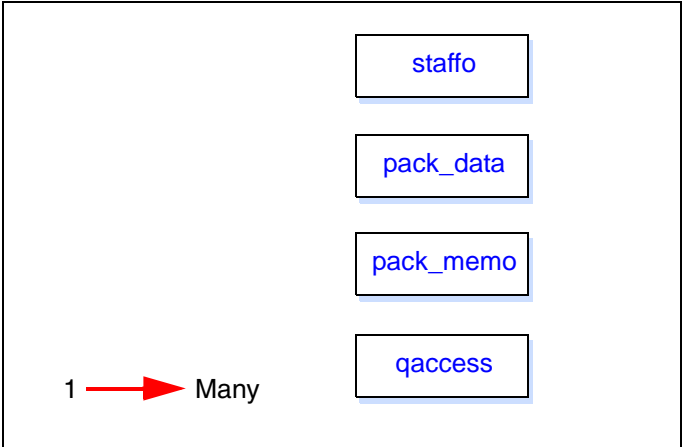
---

- [Table Relationships, page 132](#)
- [staffo, page 133](#)
- [pack\\_data, page 137](#)
- [pack\\_memo, page 139](#)
- [qaccess, page 141](#)

## Table Relationships

The following diagram shows how the tables described in this chapter 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.



## 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(
  ROWID                                NUMERIC(15)NOT NULL GENERATED ALWAYS AS
  IDENTITY (START WITH 1, INCREMENT BY 1, CACHE 50),
  o_flags                              INTEGER          ,
  o_queuename                          VARCHAR(24)       ,
  o_locker                             VARCHAR(24)       ,
  o_username                           VARCHAR(49)       ,
  o_startname                          VARCHAR(49)       ,
  o_dirname                            VARCHAR(12)      NOT NULL,
  o_dirdesc                            VARCHAR(24)       ,
  o_procname                           VARCHAR(8)NOT NULL,
  o_procdesc                           VARCHAR(24)      ,
  o_casedesc                           VARCHAR(24)       ,
  o_casenum                            NUMERIC(20)       ,
  o_placeno                             INTEGER          ,
  o_dirflags                           INTEGER          ,
  o_procflags                           INTEGER          ,
  o_host                               VARCHAR(24)      NOT NULL,
  o_pnum                               INTEGER          NOT NULL,
  o_pnumcount                           INTEGER          NOT NULL,
  o_caseptr                             NUMERIC(20)       ,
  o_reqidhost                           VARCHAR(24)      NOT NULL,
  o_reqid                               NUMERIC(20)NOT NULL,
  o_deadline                           TIMESTAMP         ,
  o_reqstamp                            TIMESTAMP         ,
  o_qparam1                             VARCHAR(24)       ,
  o_qparam2                             VARCHAR(24)       ,
  o_qparam3                             VARCHAR(12)       ,
  o_qparam4                             VARCHAR(12)       ,
  o_itempriority                        VARCHAR(24)       ,
  o_priority_changed                    TIMESTAMP         ,
  o_majorvers                           INTEGERNOT NULL,
  o_minorvers                           INTEGERNOT NULL)
```

| Column                   | Description   |
|--------------------------|---|
| <code>o_flags</code>     | Flags associated with this work item. For internal use only.  |
| <code>o_queuename</code> | Queue name of the user or group queue that contains this work item, as defined in the <a href="#">user_names</a> table. |

| Column      | Description  |
|-------------|--|
| o_locker    | Name of the user who has locked the queue (if it is locked), as defined in the <a href="#">user_names</a> table.<br><b>Note:</b> This column is not written to or updated unless the WIS_WRITELOCKS parameter in the SWDIR\etc\staffcfg file is set. |
| o_username  | Queue name of the user or group queue that contains this work item, as defined in the <a href="#">user_names</a> table.  |
| o_startname | Username of the user who started the case that this work item belongs to, as defined in the <a href="#">user_names</a> 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 <a href="#">proc_index</a> table.   |
| o_procdesc  | Procedure description of the procedure that generated this work item, as defined in the <a href="#">proc_index</a> table.  |
| o_casedesc  | Case description of the case that this work item belongs to, as defined in the <a href="#">case_information</a> table.   |
| o_casenum   | Case number of the case that this work item belongs to, as defined in the <a href="#">case_information</a> table.  |
| o_placeno   | Step mark number. For internal use only.   |
| o_dirflags  | Step flags. For internal use only.   |
| o_procfags  | Procedure flags. For internal use only.  |
| o_host      | ID of the node that this work item is associated with, as defined in the <a href="#">nodes</a> table.  |
| o_pnum      | Procedure number of the procedure that generated this work item, as defined in the <a href="#">proc_index</a> 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 <a href="#">nodes</a> table.  |

| Column             | Description  |
|--------------------|--|
| o_reqid            | Unique ID for this work item, generated from the <a href="#">sequences</a> table.  |
| o_deadline         | Date and time that the deadline (if defined) expires on this work item.<br>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     | Priority definition for this work item, in the format:<br><i>base:increment:number:period:type</i><br>where: <ul style="list-style-type: none"> <li>• <i>base</i> is the base priority value for this work item.</li> <li>• <i>increment</i> is the amount that will be added to the item's priority value whenever the <i>period</i> expires.</li> <li>• <i>number</i> is the number of increments that will be added to the item's priority value.</li> <li>• <i>period</i> is the time period, in the units specified in <i>type</i>, which must expire before the item's priority value is incremented.</li> <li>• <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.</li> </ul> |
| 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 <a href="#">proc_version</a> table.   |
| o_minorvers        | Minor version number of the procedure that generated this work item, as defined in the <a href="#">proc_version</a> table.   |

**Primary Key**      The following primary key is defined for this table.

| Key Name  | Column(s)              |
|-----------|------------------------|
| pk_staffo | o_reqid<br>o_reqidhost |

**Indexes**      The following indexes are defined for this table.

| Index Name                    | Column(s) Indexed |
|-------------------------------|-------------------|
| idx_staffo                    | o_queueName       |
| idx_staffo_rowid <sup>1</sup> | rowid             |

1. UNIQUE index

**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"><li>• a work item is kept and any changes have been made.</li><li>• a work item’s priority value changes.</li><li>• a work item is opened and the WIS_WRITELOCKS parameter in the SWDIR\etc\staffcfg file is set.</li></ul> |
| deleted     | either: <ul style="list-style-type: none"><li>• a work item is released or withdrawn.</li><li>• a case is closed or purged.</li></ul>   |

## 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` 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  |
|--------------------------|--|
| <code>reqid</code>       | ID of the work item that this field belongs to, as defined in the <code>staffo</code> table.   |
| <code>node_id</code>     | ID of the node that this field is associated with, as defined in the <code>nodes</code> table.   |
| <code>proc_id</code>     | Number of the procedure that this field belongs to, as defined in the <code>proc_index</code> table.                                       |
| <code>casenum</code>     | Case number that this field belongs to, as defined in the <code>case_information</code> table.   |
| <code>field_name</code>  | Name of the field, as defined in the <code>case_data</code> table.   |
| <code>field_value</code> | Value of the field.<br><b>Note:</b> A memo field has a value of 1. The associated memo data is stored in the <code>pack_memo</code> table. |
| <code>field_flags</code> | 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 | regid<br>node_id<br>field_name |

**Foreign Keys**      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"><li>• a step is sent out.</li><li>• a field is assigned a value on a keep or release.</li></ul>   |
| 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"><li>• a release instruction for a work item is processed by the background process.</li><li>• a work item is withdrawn.</li><li>• a case is purged.</li></ul> |

## pack\_memo

The `pack_memo` table stores memo data associated with memo fields in the `pack_data` table.

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        INTEGERNOT NULL,
  proc_id        INTEGERNOT NULL,
  casenum        NUMERIC(20)NOT NULL,
  memo_id        INTEGERNOT NULL,
  memo_index     INTEGERNOT NULL,
  memo_size      INTEGERNOT NULL,
  memo_data      BLOB(30K)NOT NULL NOT LOGGED,
  array_idx      INTEGERNOT NULL)
```

| Column                  | Description   |
|-------------------------|---|
| <code>reqid</code>      | ID of the work item that this memo belongs to, as defined in the <a href="#">staffo</a> table.  |
| <code>node_id</code>    | ID of the node that this memo is associated with, as defined in the <a href="#">nodes</a> table.  |
| <code>proc_id</code>    | Number of the procedure that this memo belongs to, as defined in the <a href="#">proc_index</a> table.  |
| <code>casenum</code>    | Case number that this memo belongs to, as defined in the <a href="#">case_information</a> 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. <ul style="list-style-type: none"> <li>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.</li> </ul> |
| <code>memo_size</code>  | Size (in bytes) of the memo data for this row.  |
| <code>memo_data</code>  | Memo data.  |

| Column    | Description   |
|-----------|---|
| array_idx | Either: <ul style="list-style-type: none"><li>• The array element index number of the memo.</li><li>• -1, if the memo is not an array memo field.</li></ul> |

**Primary Key** The following primary key is defined for this table.

| Key Name     | Column(s)   |
|--------------|---|
| pk_pack_memo | reqid<br>node_id<br>casenum<br>memo_id<br>memo_index<br>array_idx |

**Foreign Keys** 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"><li>• a step containing memo data is sent out.</li><li>• a memo field is assigned a value on a keep or release.</li></ul>  |
| 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"><li>• a release instruction for a work item containing memo data is processed by the background process.</li><li>• a work item containing memo data is withdrawn.</li><li>• a case containing memo data is purged.</li></ul> |

## 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)        )
```

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

**Foreign Keys** 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.               |

## Chapter 9

# Case Data Queue Parameters

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

## Topics

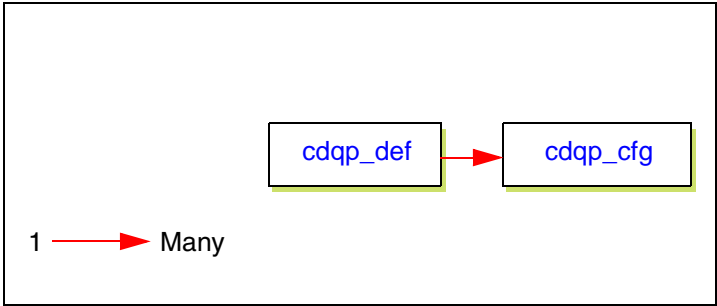
---

- [Table Relationships, page 144](#)
- [cdqp\\_def, page 145](#)
- [cdqp\\_cfg, page 147](#)

# Table Relationships

The following diagram shows how the tables described in this chapter 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.



## 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  |
|--------------------------|--|
| <code>def_id</code>      | Unique identifier for this CDQP, generated from the <a href="#">sequences</a> table.                 |
| <code>field_name</code>  | Name of the iProcess field assigned to this CDQP, as defined in the <a href="#">case_data</a> table. |
| <code>data_size</code>   | Maximum size, in characters, of this CDQP.   |
| <code>description</code> | Name used to represent this CDQP in Work Queue Manager dialogs.                                      |
| <code>is_predict</code>  | 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)           |
|--------------------------|---------------------|
| <code>pk_cdqp_def</code> | <code>def_id</code> |

**Foreign Keys** None.

**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. |

| A row is... | When...                                 |
|-------------|---|
| 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 <a href="#">sequences</a> table.   |
| <code>def_id</code>     | ID of the CDQP that is mapped to the <code>queue_name</code> queue, as defined in the <a href="#">cdqp_def</a> table.                     |
| <code>queue_name</code> | Name of the iProcess queue that the CDQP defined in <code>def_id</code> is mapped to, as defined in the <a href="#">user_names</a> table. |

**Primary Key** The following primary key is defined for this table.

| Key Name                 | Column(s)           |
|--------------------------|---------------------|
| <code>pk_cdqp_cfg</code> | <code>cfg_id</code> |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                              | Column(s)           | Referenced in Table...   |
|---------------------------------------|---------------------|--------------------------|
| <code>fk_cdqp_cfg</code> <sup>1</sup> | <code>def_id</code> | <a href="#">cdqp_def</a> |

1. This key enforces the `DELETE CASCADE` referential action.

**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. |
| updated     | never.  |
| deleted     | an existing CDQP mapping is deleted.                              |

## Chapter 10 **Queue Participation and Redirection**

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

### Topics

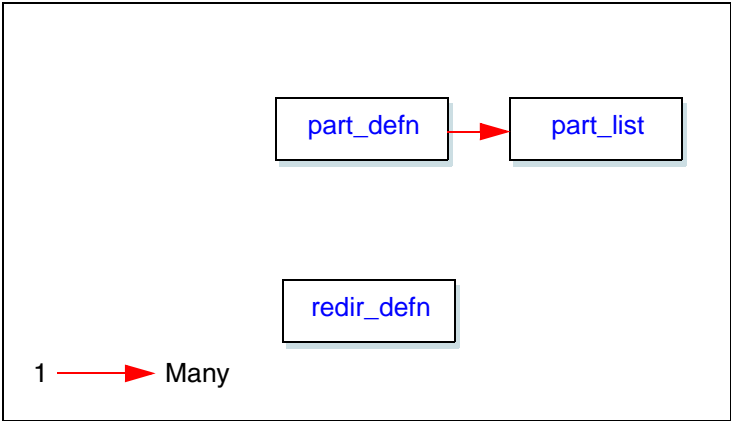
---

- [Table Relationships, page 150](#)
- [part\\_defn, page 151](#)
- [part\\_list, page 153](#)
- [redir\\_defn, page 155](#)

# Table Relationships

The following diagram shows how the tables described in this chapter 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.



## 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)     ,
  start_date       INTEGER          NOT NULL,
  end_date         INTEGER          NOT NULL)
```

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

**Primary Key** The following primary key is defined for this table.

| Key Name                  | Column(s)            |
|---------------------------|----------------------|
| <code>pk_part_defn</code> | <code>part_id</code> |

**Foreign Keys** None.

**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  |
|------------------------|--|
| <code>part_id</code>   | ID of the participation record that this participant is a member of, as defined in the <a href="#">part_defn</a> table.  |
| <code>user_name</code> | Name of the user who is allowed to participate (according to the participation definition identified by the <code>part_id</code> value), as defined in the <a href="#">user_names</a> table. |

**Primary Key** None.

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                               | Column(s)            | Referenced in Table...    |
|--|----------------------|---------------------------|
| <code>fk_part_list</code> <sup>1</sup> | <code>part_id</code> | <a href="#">part_defn</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name                    | Column(s) indexed    |
|-------------------------------|----------------------|
| <code>idx_part_list_fk</code> | <code>part_id</code> |

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

## 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   |
|--------------------------|---|
| <code>redir_id</code>    | Unique ID for this redirection record.  |
| <code>start_time</code>  | Time that this queue redirection starts.  |
| <code>start_date</code>  | Date that this queue redirection starts.  |
| <code>end_time</code>    | Time that this queue redirection ends.  |
| <code>end_date</code>    | Date that this queue redirection ends.  |
| <code>queue_name</code>  | Name of the queue from which work items are to be redirected, as defined in the <a href="#">user_names</a> table. |
| <code>destination</code> | Name of the queue to which work items are to be redirected, as defined in the <a href="#">user_names</a> table.   |

**Primary Key** The following primary key is defined for this table.

| Key Name                   | Column(s)             |
|----------------------------|-----------------------|
| <code>pk_redir_defn</code> | <code>redir_id</code> |

**Foreign Keys** 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.              |

## Chapter 11 **Administrative Tables**

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

### Topics

---

- [Table Relationships, page 158](#)
- [flag\\_table, page 159](#)
- [version, page 162](#)

# Table Relationships

---

The [flag\\_table](#) and [version](#) tables have no database-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 `dbfs_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_dbfs_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)     )
```

| Column               | Description  |
|----------------------|--|
| <code>area_id</code> | Unique ID of this area of iProcess administrative data: Either Users (1), iProcess Tables (2), Lists (3) or Roles (4). |

| Column       | Description   |
|--------------|---|
| 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"><li>an editor (for example, User Manager) when a user is editing the specified area, to prevent other users from editing the same data.</li><li>the background process while it is updating the system data, to prevent any users from editing the same data.</li></ul> |
| 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"><li>the name of the user doing the editing, or</li><li>swpro if the background process has the area locked.</li></ul>   |

**Primary Key** The following primary key is defined for this table.

| Key Name      | Column(s) |
|---------------|-----------|
| pk_flag_table | area_id   |

**Foreign Keys** 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"><li>a Move System Information operation either starts or completes.</li><li>an edit of a data area either starts or completes.</li></ul> |

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

**Foreign Keys** 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 <code>move_req</code> flag for the Users data area in the <a href="#">flag_table</a> is set to 1). |
| deleted     | never.  |

## Chapter 12 **Users and Work Queues**

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

### Topics

---

- [About User Tables, page 164](#)
- [Table Relationships, page 165](#)
- [user\\_names, page 166](#)
- [user\\_attrib, page 168](#)
- [user\\_settings, page 170](#)
- [user\\_values, page 171](#)
- [user\\_memb, page 173](#)
- [leavers, page 175](#)
- [tsys\\_user\\_names, page 177](#)
- [tsys\\_user\\_attrib, page 178](#)
- [tsys\\_user\\_values, page 179](#)
- [tsys\\_user\\_memb, page 180](#)

## 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).

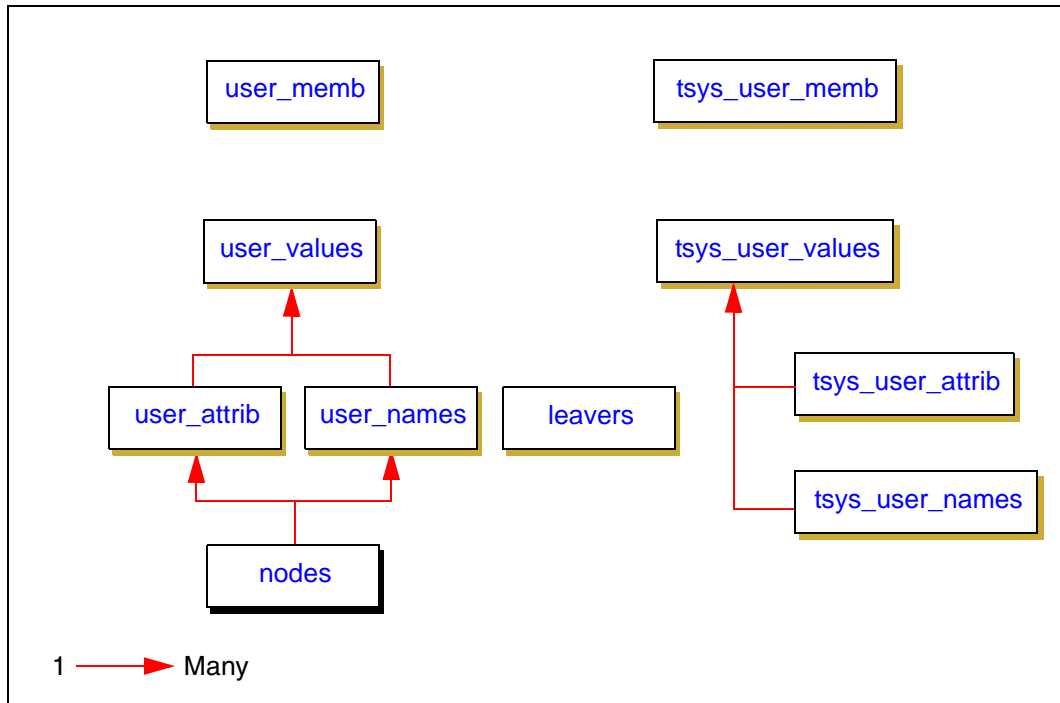
The `user_` tables are purged and rewritten with the updated information from the `tsys_user_` tables when a Move System Information is performed - *if* the [flag\\_table](#) indicates that the appropriate data area has been modified.

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 chapter 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.



## 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 <a href="#">nodes</a> table.  |
| user_id   | Unique ID for this (user or group) queue.<br><b>Note:</b> 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<br>user_type<br>node_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                   | Column(s) | Referenced in Table... |
|----------------------------|-----------|------------------------|
| fk_user_names <sup>1</sup> | node_id   | <a href="#">nodes</a>  |

1. This key enforces the DELETE CASCADE referential action.

**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 <a href="#">flag_table</a> indicates that the Users data area has been modified.</p> <p><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_user_names</a> table.</p> |
| updated     | never.  |
| deleted     | <p>a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Users data area has been modified.</p> <p><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_user_names</a> 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 <a href="#">nodes</a> 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<br>node_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                    | Column(s) | Referenced in Table... |
|-----------------------------|-----------|------------------------|
| fk_user_attrib <sup>1</sup> | node_id   | <a href="#">nodes</a>  |

1. This key enforces the DELETE CASCADE referential action.

**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 <a href="#">flag_table</a> indicates that the Users data area has been modified.</p> <p><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_user_names</a> table.</p> |
| updated     | never.  |
| deleted     | <p>a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Users data area has been modified.</p> <p><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_user_names</a> table.</p> |

## user\_settings

The `user_settings` 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_settings` table has the following structure:

```
TABLE user_settings (  
  node_id          INTEGER          NOT NULL,  
  user_name        VARCHAR(64)      NOT NULL,  
  setting_key      VARCHAR(64)      NOT NULL,  
  setting_value    VARCHAR(2048)    NOT NULL
```

| Column        | Description   |
|---------------|---|
| node_id       | ID of the node that this attribute is defined on, as defined in the <a href="#">nodes</a> table.      |
| user_name     | The name of the user whose preferences these are, as defined in the <a href="#">user_names</a> table. |
| setting_key   | The key to identify a particular setting.   |
| setting_value | The value of the particular setting identified by <code>setting_key</code> .                          |

**Primary Key** The following primary key is defined for this table.

| Key Name       | Column(s)  |
|----------------|--|
| pk_user_values | node_id<br>user_name<br>setting_key<br>setting_value |

## 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)     NOT 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 <a href="#">nodes</a> table.  |
| user_id         | ID of the (user or group) queue that this attribute value is associated with, as defined in the <a href="#">user_names</a> table.   |
| attribute_id    | ID of the attribute that this attribute value is associated with, as defined in the <a href="#">user_attrib</a> table.  |
| attribute_value | Value of this attribute.<br><b>Note:</b> 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 <a href="#">user_attrib</a> table. |
| user_type       | Type of the (user or group) queue that this attribute value is associated with, as defined in the <a href="#">user_names</a> table.   |

**Primary Key** The following primary key is defined for this table.

| Key Name       | Column(s)                                       |
|----------------|---|
| pk_user_values | user_id<br>attribute_id<br>user_type<br>node_id |

**Foreign Keys**      The following foreign keys are defined for this table.

| Key Name <sup>1</sup> | Column(s)                       | Referenced in Table... |
|-----------------------|---------------------------------|------------------------|
| fk_user_values1       | node_id<br>user_id<br>user_type | user_names             |
| fk_user_values2       | attribute_id<br>node_id         | user_attrib            |

1. These keys enforce the DELETE CASCADE referential action.

**Indexes**      The following indexes are defined for this table.

| Index Name      | Column(s) Indexed               |
|-----------------|---------------------------------|
| idx_usrval_fk1  | node_id<br>user_id<br>user_type |
| idx_usrval_fk2  | attribute_id<br>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       | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Users data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_user_names</a> table. |
| updated     | never.   |
| deleted     | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Users data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_user_names</a> 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 <a href="#">nodes</a> table. |
| user_id  | ID of the user who belongs to the group, as defined in the <a href="#">user_names</a> table.                  |
| group_id | ID of the group that the user belongs to, as defined in the <a href="#">user_names</a> table.                 |

**Primary Key** The following primary key is defined for this table.

| Key Name     | Column(s)                      |
|--------------|--------------------------------|
| pk_user_memb | user_id<br>node_id<br>group_id |

**Foreign Keys** None.

**Indexes** The following index is defined for this table.

| Index Name    | Column(s) Indexed   |
|---------------|---------------------|
| idx_user_memb | user_id<br>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       | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Users data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_user_names</a> table. |
| updated     | never.   |
| deleted     | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Users data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_user_names</a> 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 <a href="#">nodes</a> 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"> <li>0 (LEAVER_WILL_BE_REDIRECTED) The leaver will be redirected.</li> <li>1 (LEAVER_IS_BEING_REDIRECTED) The leaver is being redirected.</li> <li>2 (LEAVER_FINISH_REDIRECTION) The leaver has been redirected.</li> </ul> |

**Primary Key** The following primary key is defined for this table.

| Key Name   | Column(s)            |
|------------|----------------------|
| pk_leavers | user_name<br>node_id |

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

## **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 foreign keys or indexes are defined.
- 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 foreign keys or indexes are defined.
- 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_value`.
- The foreign key names are `fk_tsysusrval_fk1` and `fk_tsysusrval_fk2`.
- The index names are `idx_tsysusrval_fk1` and `idx_tsysusrval_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.)

## Chapter 13 **Roles**

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

### Topics

---

- [About Roles, page 182](#)
- [Table Relationships, page 183](#)
- [role\\_users, page 184](#)
- [tsys\\_role\\_users, page 186](#)

## 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).

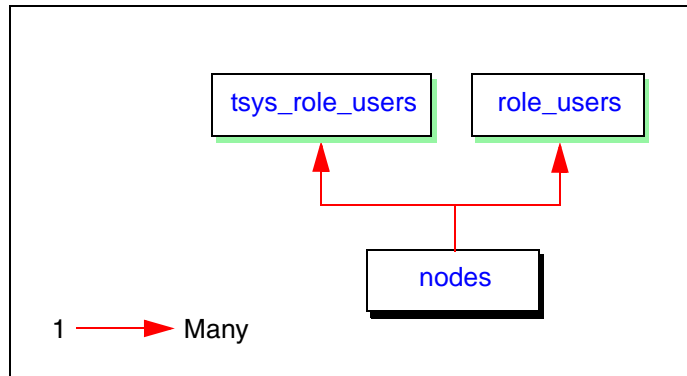
When a Move System Information is performed, if the [tsys\\_role\\_users](#) table has been changed, the [role\\_users](#) table is purged and rewritten with the updated information from the [tsys\\_role\\_users](#) table.

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 chapter 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.



# 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 <a href="#">nodes</a> 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 <a href="#">user_names</a> table. |

**Primary Key** The following primary key is defined for this table.

| Key Name      | Column(s)          |
|---------------|--------------------|
| pk_role_users | role_id<br>node_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                   | Column(s) | Referenced in Table... |
|----------------------------|-----------|------------------------|
| fk_role_users <sup>1</sup> | node_id   | <a href="#">nodes</a>  |

1. This key enforces the DELETE CASCADE referential action.

**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 <a href="#">flag_table</a> indicates that the Roles data area has been modified.</p> <p><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_role_users</a> table.</p> |
| updated     | never.  |
| deleted     | <p>a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Roles data area has been modified.</p> <p><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_role_users</a> table.</p> |

## 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 foreign key name is `fk_tsys_role_users`.
- The index name is `idx_tsysroleusr_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.)

## Chapter 14 TIBCO iProcess Tables

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



This chapter uses the term *TIBCO iProcess table* to mean an iProcess table, and *table* to mean a DB2 table.

### Topics

---

- [About TIBCO iProcess Tables, page 188](#)
- [Table Relationships, page 189](#)
- [dbs\\_names, page 190](#)
- [dbs\\_fields, page 192](#)
- [dbs\\_values, page 194](#)
- [tsys\\_dbs\\_names, page 196](#)
- [tsys\\_dbs\\_fields, page 197](#)
- [tsys\\_dbs\\_values, page 198](#)
- [str\\_dbs\\_names, page 199](#)
- [str\\_dbs\\_fields, page 200](#)
- [ttmp\\_dbs\\_names, page 201](#)
- [ttmp\\_dbs\\_fields, page 202](#)
- [ttmp\\_dbs\\_values, page 203](#)

## 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.<br><b>Note:</b> 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).

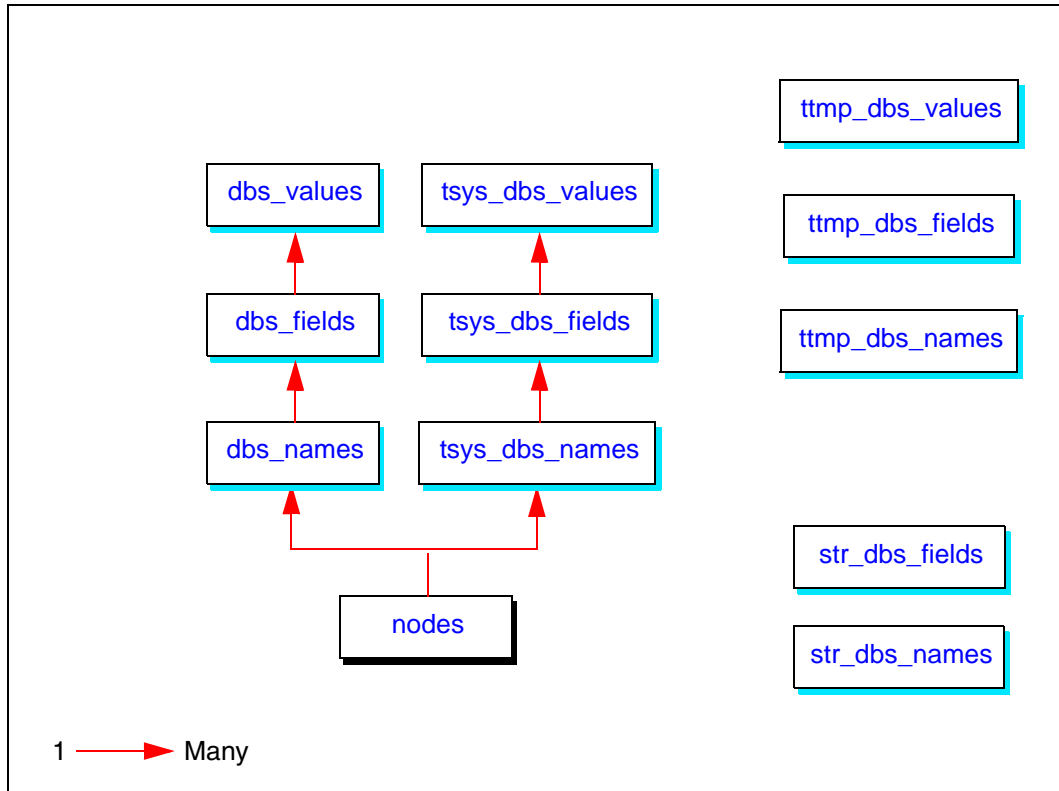
When a Move System Information is performed, if the tsys\_db\_ tables have been changed, the db\_ and/or str\_db\_ tables are purged and rewritten with the updated information from the tsys\_db\_ tables.

Access to the db\_ and tsys\_db\_ 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 chapter 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.



# dbs\_names

The `dbs_names` table holds the names of all *installed* TIBCO iProcess Engine tables.

**Structure** The `dbs_names` table has the following structure:

```
TABLE dbs_names (  
    node_id            INTEGER            NOT NULL,  
    dbs_id             INTEGER            NOT NULL,  
    dbs_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 <a href="#">nodes</a> table. |
| dbs_id   | Unique ID for this iProcess Engine table.  |
| dbs_name | Name of this iProcess Engine table.  |

**Primary Key** The following primary key is defined for this table.

| Key Name     | Column(s)         |
|--------------|-------------------|
| pk_dbs_names | dbs_id<br>node_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                  | Column(s) | Referenced in Table... |
|---------------------------|-----------|------------------------|
| fk_dbs_names <sup>1</sup> | node_id   | <a href="#">nodes</a>  |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name       | Column(s) Indexed |
|------------------|-------------------|
| idx_dbs_names_fk | node_id           |

**Table Activity** The `dbcs_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       | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Tables data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_dbs_names</a> table. |
| updated     | never.   |
| deleted     | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Tables data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_dbs_names</a> table. |

## dbfs\_fields

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

**Structure** The `dbfs_fields` table has the following structure:

```
TABLE dbfs_fields (  
    node_id          INTEGER          NOT NULL,  
    dbfs_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  |
|----------------|--|
| node_id        | ID of the node that this field is defined on, as defined in the <a href="#">nodes</a> table.       |
| dbfs_id        | ID of the table that this field is defined in, as defined in the <a href="#">dbfs_names</a> table. |
| field_id       | Unique ID for the field in this TIBCO iProcess Engine table.                                       |
| field_name     | Name of this field.  |
| field_type     | Field type: Either ASCII (A), Numeric (R), Date (D) or Time (T).                                   |
| field_length   | Length of this field, in characters.   |
| field_decimals | 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_dbfs_fields | dbfs_id<br>field_id<br>node_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                    | Column(s)          | Referenced in Table...     |
|-----------------------------|--------------------|----------------------------|
| fk_dbfs_fields <sup>1</sup> | node_id<br>dbfs_id | <a href="#">dbfs_names</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following indexes are defined for this table.

| Index Name        | Column(s) Indexed  |
|-------------------|--------------------|
| idx_dbs_fields_fk | dbs_id<br>node_id  |
| idx_dbs_fields    | field_id<br>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 <a href="#">flag_table</a> indicates that the Tables data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_dbs_fields</a> table. |
| updated     | never.  |
| deleted     | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Tables data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_dbs_fields</a> table. |

## db\_values

The db\_values table holds all field values for all *installed* TIBCO iProcess Engine tables.

**Structure** The db\_values table has the following structure:

```
TABLE db_values (
  node_id          INTEGER          NOT NULL,
  db_id            INTEGER          NOT NULL,
  record_id        INTEGER          NOT NULL,
  field_id         INTEGER          NOT NULL,
  field_value      VARCHAR(30)      )
```

| Column      | Description   |
|-------------|---|
| node_id     | ID of the node that this field value is stored on, as defined in the <a href="#">nodes</a> table.     |
| db_id       | ID of the table that this field value is stored in, as defined in the <a href="#">db_names</a> table. |
| record_id   | Unique ID for this record in the iProcess Engine table.   |
| field_id    | ID of the field held in this record, as defined in the <a href="#">db_fields</a> 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_db_values | db_id<br>record_id<br>field_id<br>node_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                  | Column(s)                    | Referenced in Table...    |
|---------------------------|------------------------------|---------------------------|
| fk_db_values <sup>1</sup> | db_id<br>field_id<br>node_id | <a href="#">db_fields</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following indexes are defined for this table.

| Index Name        | Column(s) Indexed               |
|-------------------|---------------------------------|
| idx_dbs_values_fk | dbs_id<br>field_id<br>node_id   |
| idx_dbs_values    | record_id<br>field_id<br>dbs_id |

**Table Activity** The dbs\_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       | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Tables data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_dbs_values</a> table. |
| updated     | never.  |
| deleted     | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Tables data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_dbs_values</a> table. |

## `tsys_dbs_names`

---

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

- The primary key name is `pk_tsys_dbs_names`.
- The foreign key name is `fk_tsys_dbs_names`.
- The index name is `idx_tsysdbsname_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 foreign key name is `fk_tsys_dbs_fields`.
- The index names are `idx_tsysdbsfld_fk` and `idx_tsys_dbs_field`.
- 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 foreign key name is `fk_tsys_dbs_values`.
- The index names are `idx_tsysdbsval_fk` and `idx_tsys_dbs_value`.
- 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 foreign keys or 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`.
- No foreign keys are defined.
- 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 foreign keys or indexes are defined.
- In most situations the number of rows in the table should be zero.

## tmp\_dbs\_fields

---

The tmp\_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\_tmp\_dbs\_fields.
- No foreign keys or 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 foreign keys or indexes are defined.
- In most situations the number of rows in the table should be zero.



## Chapter 15   **Lists**

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

### Topics

---

- [About Lists, page 206](#)
- [Table Relationships, page 207](#)
- [list\\_names, page 208](#)
- [list\\_values, page 210](#)
- [tsys\\_list\\_names, page 212](#)
- [tsys\\_list\\_values, page 213](#)
- [ttmp\\_list\\_names, page 214](#)
- [ttmp\\_list\\_values, page 215](#)

# 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).

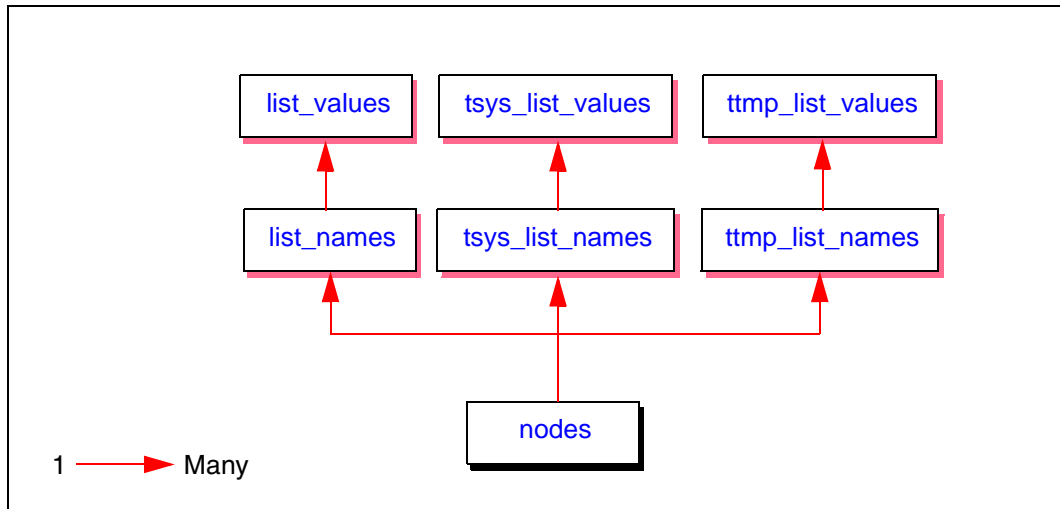
The `list_` tables are purged and rewritten with the updated information from the `tsys_list_` tables when a Move System Information is performed - *if* the [flag\\_table](#) indicates that the Lists data area has been modified

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 chapter 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.



# 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 <a href="#">nodes</a> 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<br>node_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                   | Column(s) | Referenced in Table... |
|----------------------------|-----------|------------------------|
| fk_list_names <sup>1</sup> | node_id   | <a href="#">nodes</a>  |

1. This key enforces the DELETE CASCADE referential action.

**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       | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Lists data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_list_names</a> table. |
| updated     | never.   |
| deleted     | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Lists data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_list_names</a> table. |

## 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 <a href="#">nodes</a> table.      |
| list_id    | ID of the list that this list item is stored in, as defined in the <a href="#">list_names</a> 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<br>record_id<br>node_id |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                    | Column(s)          | Referenced in Table...     |
|-----------------------------|--------------------|----------------------------|
| fk_list_values <sup>1</sup> | node_id<br>list_id | <a href="#">list_names</a> |

1. This key enforces the DELETE CASCADE referential action.

**Indexes** The following index is defined for this table.

| Index Name        | Column(s) Indexed  |
|-------------------|--------------------|
| idx_list_value_fk | list_id<br>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       | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Lists data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_list_values</a> table. |
| updated     | never.  |
| deleted     | a Move System Information is performed, if the <a href="#">flag_table</a> indicates that the Lists data area has been modified.<br><b>Note:</b> The table is purged and rewritten using the values from the <a href="#">tsys_list_values</a> table. |

## `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 foreign key name is `fk_tsys_list_names`.
- The index name is `idx_tsyslstname_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_value`.
- The foreign key name is `fk_tsys_list_value`.
- The index name is `idx_tsyslstvals_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.)

## tmp\_list\_names

---

The tmp\_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\_tmp\_list\_names.
- The foreign key name is fk\_tmp\_list\_names.
- The index name is idx\_tmplstname\_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_value`.
- `fk_ttmp_list_value`The index name is `idx_ttmpilstvals_fk`.
- In most situations the number of rows in the table should be zero.



## Chapter 16 **iProcess Server Plug-ins**

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

### Topics

---

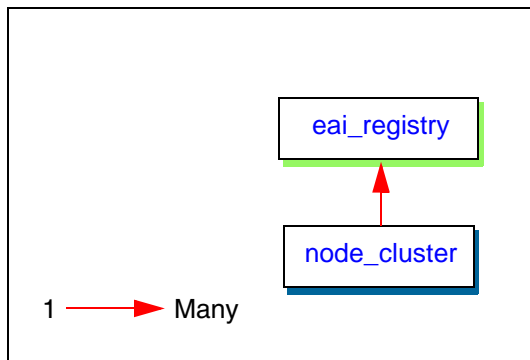
- [Table Relationships, page 218](#)
- [eai\\_registry, page 219](#)

## Table Relationships

---

The following diagram shows how the tables described in this chapter 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.



## 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)        )
```

| Column                          | Description   |
|---------------------------------|---|
| <code>eai_type</code>           | Short name for the EAI Step type that this iProcess server plug-in supports. For example, one of the following: <ul style="list-style-type: none"> <li>• <code>EAIDB</code> EAI Database</li> <li>• <code>EATSCR</code> EAI Script</li> <li>• <code>EAIWEBSERVICES</code> EAI Web Services</li> </ul>             |
| <code>logical_machine_id</code> | ID of the computer that this iProcess server plug-in is installed on, as defined in the <a href="#">node_cluster</a> table.<br><b>Note:</b> 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. |
| <code>release_version</code>    | Version number of this iProcess server plug-in (for example, <code>i10.0-u(4.0)</code> ).   |
| <code>plugin_library</code>     | Pathname (on this <code>logical_machine_id</code> ) where this EAI server plug-in is installed.   |
| <code>init_params</code>        | Startup parameters used by this iProcess server plug-in.  |

**Primary Key** The following primary key is defined for this table.

| Key Name                     | Column(s)  |
|------------------------------|--|
| <code>pk_eai_registry</code> | <code>eai_type</code><br><code>logical_machine_id</code> |

**Foreign Keys**      The following foreign key is defined for this table.

| Key Name                     | Column(s)          | Referenced in Table... |
|------------------------------|--------------------|------------------------|
| fk_eai_registry <sup>1</sup> | logical_machine_id | node_cluster           |

1. This key enforces the DELETE CASCADE referential action.

**Indexes**      The following index is defined for this table.

| Index Name         | Column(s) Indexed  |
|--------------------|--------------------|
| idx_eairegistry_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.             |

## Chapter 17 Firewall Port Ranges

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



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*.

### Topics

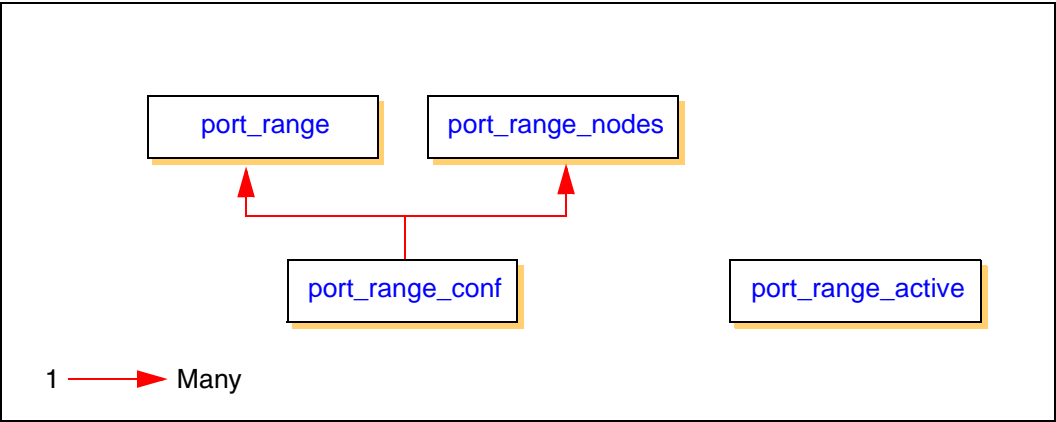
---

- [Table Relationships, page 222](#)
- [port\\_range, page 223](#)
- [port\\_range\\_active, page 225](#)
- [port\\_range\\_conf, page 227](#)
- [port\\_range\\_nodes, page 229](#)

## Table Relationships

The following diagram shows how the tables described in this chapter 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.



## 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 <a href="#">port_range_conf</a> 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>             | Defines whether this port/RPC number is available or in use by a process. One of the following values: <ul style="list-style-type: none"> <li>• -2 Reserved for future use.</li> <li>• -1 Unobtainable. (A process tried to use the port but found that it was already in use.)</li> <li>• 0 Unallocated.</li> <li>• 1 Allocated to the process defined by the <code>logical_machine_id</code>, <code>logical_process_name</code> and <code>logical_process_instance</code> columns.</li> </ul> |
| <code>logical_machine_id</code> | Either: <ul style="list-style-type: none"> <li>• ID of the server where the process instance that this port/RPC number has been allocated to runs, as defined in the <a href="#">node_cluster</a> table.</li> <li>• 0, if the port/RPC number has not been allocated to a process.</li> </ul>   |

| Column                   | Description   |
|--------------------------|---|
| 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 |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name      | Column(s)     | Referenced in Table...          |
|---------------|---------------|---------------------------------|
| fk_port_range | port_range_id | <a href="#">port_range_conf</a> |

**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 <a href="#">port_range_conf</a> 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 <a href="#">port_range_conf</a> 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.



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  |
|---------------------------------------|--|
| <code>logical_machine_id</code>       | ID of the server where this process instance runs, as defined in the <a href="#">node_cluster</a> table.   |
| <code>logical_process_name</code>     | Logical name of this process instance.<br><b>Note:</b> 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> | Unique ID for this process instance.   |
| <code>process_id</code>               | Operating system process ID of this process instance.  |
| <code>port_number</code>              | Port number being used by this process instance.   |
| <code>rpc_number</code>               | RPC number being used by this process instance.  |

### Primary Key

The following primary key is defined for this table.

| Key Name                          | Column(s)   |
|-----------------------------------|---|
| <code>pk_port_range_active</code> | <code>logical_machine_id</code> ,<br><code>logical_process_name</code> ,<br><code>logical_process_instance</code> |

### Foreign Keys

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 <a href="#">port_range</a> 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.



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   |
|----------------------------|---|
| <code>port_range_id</code> | Unique ID of this particular port range configuration.  |
| <code>range_mode</code>    | Mode used by this port range configuration. One of the following values: <ul style="list-style-type: none"> <li>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.</li> <li>1 Port ranging. A process uses a port number allocated from within the defined range, and an RPC number based on the process ID.</li> <li>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.</li> <li>3 Port and RPC ranging. A process uses both a port number and an RPC number allocated from within the defined ranges.</li> </ul> |
| <code>range_size</code>    | The number of port and RPC numbers allowed in the port number and RPC number ranges.  |
| <code>port_start</code>    | The first number in the defined range of port numbers. (The last number = <code>port_start + range_size</code> .)   |
| <code>rpc_start</code>     | The first number in the defined range of RPC numbers. (The last number = <code>rpc_start + range_size</code> .)   |

**Primary Key**      The following primary key is defined for this table.

| Key Name           | Column(s)     |
|--------------------|---------------|
| pk_port_range_conf | port_range_id |

**Foreign Keys**      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 <i>SWDIR\util\swadm</i> 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 <i>SWDIR\util\swadm</i> utility. |
| deleted     | a user deletes an existing port range configuration using the <i>SWDIR\util\swadm</i> 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).



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   |
|---------------------------------|---|
| <code>port_range_id</code>      | ID of a particular port range configuration, as defined in the <a href="#">port_range_conf</a> table.       |
| <code>logical_machine_id</code> | ID of the server using this port range configuration, as defined in the <a href="#">node_cluster</a> table. |

**Primary Key** The following primary key is defined for this table.

| Key Name                         | Column(s)   |
|----------------------------------|---|
| <code>pk_port_range_nodes</code> | <code>port_range_id</code> ,<br><code>logical_machine_id</code> |

**Foreign Keys** The following foreign key is defined for this table.

| Key Name                         | Column(s)                  | Referenced in Table...          |
|----------------------------------|----------------------------|---------------------------------|
| <code>fk_port_range_nodes</code> | <code>port_range_id</code> | <a href="#">port_range_conf</a> |

**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 <code>SWDIR\util\swadm</code> utility.      |
| updated     | a user moves a machine from sitting behind one particular port range configuration to another, using the <code>SWDIR\util\swadm</code> utility.           |
| deleted     | a user removes a machine from the list of servers that sit behind a particular port range configuration, using the <code>SWDIR\util\swadm</code> utility. |

## Chapter 18 **WQS/WIS Shared Memory**

This chapter describes the [wqs\\_index](#) table.

# Table Relationships

---

The [wqs\\_index](#) table has no database-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   |
|---------------------------------------|---|
| <code>logical_machine_id</code>       | ID of the server where the WIS process that is handling this work queue is running, as defined in the <a href="#">node_cluster</a> table.   |
| <code>logical_process_instance</code> | ID of the instance of the WIS process that is handling this work queue, as defined in the <a href="#">process_config</a> table.   |
| <code>queue_name</code>               | Name of the work queue.   |
| <code>total_items</code>              | Total number of items in this work queue.<br><b>Note:</b> 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. |
| <code>last_cache_time</code>          | Either: <ul style="list-style-type: none"> <li>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.</li> <li>-1, if the work queue has not yet been cached.</li> </ul>   |

| Column Name                   | Description  |
|-------------------------------|--|
| <code>new_items</code>        | Number of new, unopened items in this work queue.  |
| <code>deadline_items</code>   | Number of items in this work queue that have deadlines.  |
| <code>urgent_items</code>     | Number of items in this work queue that have an urgent priority.   |
| <code>redir_queue_name</code> | Either: <ul style="list-style-type: none"> <li>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>).</li> <li>empty, if the queue is currently not being redirected (<code>is_redirected = 0</code>).</li> </ul> |
| <code>is_cached</code>        | Indicates whether the queue is currently cached by the WIS process. Either: <ul style="list-style-type: none"> <li>1, if the queue is cached.</li> <li>0, if the queue is not cached.</li> </ul>   |
| <code>is_group</code>         | Indicates whether the queue is a Group queue. Either: <ul style="list-style-type: none"> <li>1, if the queue is a Group queue.</li> <li>0, if the queue is a User queue.</li> </ul>  |
| <code>is_test</code>          | Indicates whether the queue is a Test queue. Either: <ul style="list-style-type: none"> <li>1, if the queue is a Test queue.</li> <li>0, otherwise.</li> </ul>   |
| <code>is_redirected</code>    | Indicates whether the queue is currently being redirected to <code>redir_queue_name</code> . Either: <ul style="list-style-type: none"> <li>1, if the queue is currently redirected.</li> <li>0, otherwise.</li> </ul>   |
| <code>is_disabled</code>      | Indicates whether the queue is disabled. Either: <ul style="list-style-type: none"> <li>1, if the queue is currently disabled.</li> <li>0, otherwise.</li> </ul>   |

**Primary Key** The following primary key is defined for this table.

| Key Name     | Column(s)             |
|--------------|-----------------------|
| pk_wqs_index | queue_name<br>is_test |

**Foreign Keys** 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"> <li>an existing work queue is re-allocated to a different WIS process by the WQS process.</li> <li>a MOVESYSINFO has been processed by the WQS process.</li> <li>the update thread in the WQS process writes the contents of the WQS/WIS shared memory to the database. This update occurs every WQS_PERSIST_SHMEM seconds.</li> </ul> <p><b>Note:</b> See "Administering Process Attributes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for more information about the WQS_PERSIST_SHMEM 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.   |



## Appendix A Views

The following database views are defined for internal use:

- `dbs_nm_fld`
- `tsys_dbs_nm_fld`
- `ttmp_dbs_nm_fld`
- `str_dbs_nm_fld`
- `lst_nm_val`
- `tsys_lst_nm_val`
- `ttmp_lst_nm_val`



For more information about these views please see the database creation script (`init2Kdb2.sql`).



## Appendix B **SSOLite Stored Procedures**

This appendix describes the SSOLite stored procedures.

### Topics

---

- [Overview, page 240](#)
- [Using SSOLite Stored Procedures, page 241](#)
- [Data Procedures, page 247](#)
- [Command Procedures, page 254](#)
- [Control Procedures, page 277](#)
- [Debug Procedures, page 287](#)

## 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.

## 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, page 241](#)
- [Transactional Processing, page 241](#)
- [Post-installation Procedures, page 242](#)
- [Handling Exceptions, page 242](#)
- [Processing Queues, page 244](#)
- [Prioritizing Messages, page 245](#)

### 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.

## Post-installation Procedures

The installation/upgrade process installs a shared library stored procedure, SW\_SEQ\_TRANS, that is used by the SSOLite stored procedures for getting sequence numbers outside of the current transaction. This is installed in the following folder:

```
~$DB2INSTANCE/sqlllib/function
```

Since the external stored procedure, SW\_SEQ\_TRANS, is fenced, you must grant the "fenced" user (reserved for executing stored procedures) permissions on the [sequences](#) table. For example:

```
GRANT REFERENCES, SELECT, INSERT, DELETE, UPDATE ON swpro.sequences TO USER DB2FENC2
```

## Handling Exceptions

SSOLite stored procedures signal a SQL state and set the error text accordingly 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.

For example,

```
SIGNAL SQLSTATE '38200' SET MESSAGE_TEXT = 'SW_GRAFTCOUNT:(SWERROR) Failed to get node ID (ID:001024)';
```

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.



Some of the stored procedures listed in the table are not described in this chapter. These are lower level stored procedures that may be called by some or all of the stored procedures that are described in this chapter.

| Stored Procedure    | SQLSTATE Value | Error Text   |
|---------------------|----------------|--|
| SW_GET_NODE_DETAILS | 38200          | MBox Queue Name(s) not found in database (ID:001008) |

| Stored Procedure   | SQLSTATE Value | Error Text   |
|--------------------|----------------|--|
| SW_GET_PROCEDURE   | 38200          | Procedure details not found in database for procedure name= <i>proc_name</i> (ID:001010)                               |
|                    | 38200          | Procedure version not found in database for procedure name= <i>proc_name</i> , Case Num= <i>case_num</i> (ID:001011)   |
|                    | 38200          | Latest Released or Unreleased Procedure version not found in database for procedure name= <i>proc_name</i> (ID:001012) |
|                    | 38200          | Procedure version not found in database for procedure name= <i>proc_name major_version minor_version</i> (ID:001013)   |
| SW_SUSPEND         | 38200          | Suspend type ( <i>suspend_type</i> ) is invalid, expected 2 (suspend) or 0 (activate) (ID:001014)                      |
|                    | 38200          | Failed to get node ID (ID:001015)  |
|                    | 38200          | Failed to get procedure ID (ID:001016)   |
|                    | 38200          | Failed to get request ID (ID:001017)   |
|                    | 38200          | Case ( <i>case_num</i> ) is already active (ID:001036)   |
|                    | 38200          | Case ( <i>case_num</i> ) is dead (ID:001037)   |
|                    | 38200          | Case ( <i>case_num</i> ) is already suspended (ID:001038)  |
|                    | 38200          | Procedure and case information does not match (ID:001043)  |
| SW_GETCASE_STATUS  | 38200          | Failed to find case information for case: <i>case_num</i> (ID:001019)  |
| SW_DELAYED_RELEASE | 38200          | Failed to get node ID (ID:001022)  |
|                    | 38200          | Failed to get procedure ID (ID:001023)   |
| SW_GRAFTCOUNT      | 38200          | Failed to get node ID (ID:001024)  |
|                    | 38200          | Failed to get procedure ID (ID:001025)   |
|                    | 38200          | Failed to get request ID (ID:001026)   |

| Stored Procedure | SQLSTATE Value | Error Text  |
|------------------|----------------|---|
| SW_EVENT         | 38200          | Failed to get node ID (ID:001027)                         |
|                  | 38200          | Failed to get procedure ID (ID:001028)                    |
|                  | 38200          | Failed to get request ID (ID:001029)                      |
| SW_PURGE         | 38200          | Failed to get node ID (ID:001030)                         |
|                  | 38200          | Failed to get procedure ID (ID:001031)                    |
|                  | 38200          | Failed to get request ID (ID:001032)                      |
|                  | 38200          | Procedure and case information does not match (ID:001041) |
| SW_CLOSE         | 38200          | Failed to get node ID (ID:001033)                         |
|                  | 38200          | Failed to get procedure ID (ID:001034)                    |
|                  | 38200          | Failed to get request ID (ID:001035)                      |
|                  | 38200          | Case ( <i>case_num</i> ) is dead (ID:001039)              |
|                  | 38200          | Procedure and case information does not match (ID:001042) |

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 (BGMBBOX1) is used first. As a result, BGMBBOX1 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
```

---



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 on page 23.](#)
- "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.



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*.



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 wish 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.

```
CALL owner.SW_ADD_PACK_DATA ('F1', 'DataItem1')/
CALL owner.SW_ADD_PACK_DATA ('F2', 'DataItem2')/
CALL owner.SW_CASESTART ('CUSTREQ', -1, -1, 'Case1', 'user35', '', '?', ?)/
CALL owner.SW_CASESTART ('CUSTREQ', -1, -1, 'Case2', 'user35', '', '?', ?)/
```

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 below.

```
CALL owner.SW_ADD_PACK_DATA ('F1', 'DataItem1')/  
CALL owner.SW_ADD_PACK_DATA ('F2', 'DataItem2')/  
CALL owner.SW_CASESTART ('CUSTREQ', -1, -1, 'Case1', 'user35', '', '?', ?)/  
CALL owner.SW_ADD_PACK_DATA ('F1', 'DataItem1')/  
CALL owner.SW_ADD_PACK_DATA ('F2', 'DataItem2')/  
CALL owner.SW_CASESTART ('CUSTREQ', -1, -1, 'Case2', 'user35', '', '?', ?)/
```

---

## 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.

|               |                    |                |
|---------------|--------------------|----------------|
| <b>Syntax</b> | SW_ADD_PACK_MEMO ( |                |
|               | memo_name          | varchar(31),   |
|               | memo_length        | integer,       |
|               | memo_data          | varchar(2048), |
|               | array_idx          | integer)       |

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 wish 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.

```
CALL owner.SW_ADD_PACK_MEMO ('F1', 'MemoDataItem1', 0)/
CALL owner.SW_ADD_PACK_MEMO ('F2', 'MemoDataItem2', 0)/
CALL owner.SW_CASESTART ('CUSTREQ', -1, -1, 'Case1', 'user35', '', '?', ?)/
```

## 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 below.
- `proc_min_ver` is either the minor version number of the `proc_name` procedure, or -1. See the notes below.
- `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).



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.

---

```
CALL swpro.SW_ADD_PACK_DATA('TEXT1', 'New customer name')/  
CALL 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\\_EVENT](#)
- [SW\\_EVENT\\_UPDATE\\_PACK](#)
- [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>    | varchar(8),   |
| <i>proc_maj_ver</i> | integer,      |
| <i>proc_min_ver</i> | integer,      |
| <i>case_num</i>     | numeric(20),  |
| <i>Audit_id</i>     | integer       |
| <i>Audit_step</i>   | varchar(8)    |
| <i>Audit_desc</i>   | varchar(24)   |
| <i>User_id</i>      | varchar(255)) |

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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.
- *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).



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.

---

```
CALL swpro.SW_AUDIT ('CARPOOL', -1, -1, 53, 131, '', 'BW Activity', 'BW User')/
```

---

## SW\_CASEREOPEN

---

The SW\_CASEREOPEN procedure resurrects a case.

**Syntax**

|                  |              |
|------------------|--------------|
| SW_CASEREOPEN (  |              |
| <i>proc_name</i> | varchar(8),  |
| <i>user_id</i>   | varchar(24), |
| <i>step_name</i> | varchar(8),  |
| <i>case_num</i>  | 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.

---

```
CALL ssolite.SW_CASEREOPEN ('CUSTREQ', 'user35', 'STEP1', 101)/
```

---

# SW\_CASESTART

---

The SW\_CASESTART procedure starts a case of a procedure.

|               |                     |               |
|---------------|---------------------|---------------|
| <b>Syntax</b> | SW_CASESTART (      |               |
|               | <i>proc_name</i>    | varchar(8) ,  |
|               | <i>proc_maj_ver</i> | integer ,     |
|               | <i>proc_min_ver</i> | integer ,     |
|               | <i>case_desc</i>    | varchar(24) , |
|               | <i>user_id</i>      | varchar(24) , |
|               | <i>step_name</i>    | varchar(8) ,  |
|               | <i>case_num</i>     | numeric(20) , |
|               | <i>request_id</i>   | numeric(20))  |

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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.
- *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.

**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 current precedence settings defined for the user who is starting the case (*user\_id*) or, if these are not defined,
2. the latest released version of the procedure or, if no released version exists,

3. the latest unreleased version of the procedure.



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.

---

```
CALL owner.SW_ADD_PACK_DATA ('CustName', 'Allsop, J.A')/
CALL owner.SW_ADD_PACK_DATA ('CustID', '478163')/
CALL owner.SW_CASESTART ('CUSTREQ', -1, -1, 'Refund request', 'user35', '', '?', ?)/
```

---

# SW\_CLOSE

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 (  
    *proc\_name*                    varchar(8),  
    *proc\_maj\_ver*                integer,  
    *proc\_min\_ver*                integer,  
    *case\_number*                numeric(20),  
    *user\_id*                    varchar(24))

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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).



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.

```
CALL owner.SW_CLOSE ('CUSTREQ', -1, -1, 103, 'user35')/
```

## 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**

|                                       |                           |
|---------------------------------------|---------------------------|
| <code>SW_CLOSE_WITHOUT_EVENT (</code> |                           |
| <code>    proc_name</code>            | <code>varchar(8),</code>  |
| <code>    proc_maj_ver</code>         | <code>integer,</code>     |
| <code>    proc_min_ver</code>         | <code>integer,</code>     |
| <code>    case_number</code>          | <code>numeric(20),</code> |
| <code>    user_id</code>              | <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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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).



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.

---

```
CALL owner.SW_CLOSE_WITHOUT_EVENT ('CUSTREQ', -1, -1, 103, 'user35')/
```

---

# SW\_EVENT

---

The SW\_EVENT procedure triggers a specific event on a case of a procedure.

Syntax

```
SW_EVENT (  
    proc_name          varchar(8) ,  
    proc_maj_ver       integer ,  
    proc_min_ver       integer ,  
    step_name          varchar(8) ,  
    case_num           numeric(20) ,  
    user_id            varchar(24))
```

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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).



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.

```
CALL owner.SW_EVENT ('CUSTREQ', -1, -1, 'STEP1', 101, 'swadmin')/
```

---

## SW\_EVENT\_UPDATE\_PACK

---

The `SW_EVENT_UPDATE_PACK` procedure is the same as [SW\\_EVENT](#), 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 (
    proc_name          varchar(8),
    proc_maj_ver       integer,
    proc_min_ver       integer,
    step_name          varchar(8),
    case_num           numeric(20),
    user_id            varchar(24))
```

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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).



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.

---

```
CALL owner.SW_EVENT ('CUSTREQ', -1, -1, 'STEP1', 101, 'swadmin')/
```

---

# 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), |
| <i>proc_type</i>    | varchar(10), |
| <i>case_started</i> | timestamp)   |

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.

---

```
CALL owner.SW_GETCASE_STATUS (8, ?, ?, ?)/
```

---

This results in output displaying the status of the case (as well as the procedure type and date and time the procedure was started). For example:

---

```
Active
Main
MAY 25 2005 3:36PM
```

---

## 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 (
    proc_name          varchar(8),
    proc_maj_ver       integer,
    proc_min_ver       integer,
    case_number        numeric(20),
    graft_step_name     varchar(8),
    graft_proc_name     varchar(8),
    graft_proc_maj_ver  integer,
    graft_proc_min_ver  integer,
    graft_id            varchar(49))
```

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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 below
- *graft\_proc\_min\_ver* is either the minor version number of the *graft\_proc\_name* procedure, or -1. See the notes below
- *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).



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.

---

```
CALL owner.SW_GRAFT ('CUSTREQ', -1, -1, 101, 'GRAFT01', 'SUBPROC1', -1, -1,
'UNIQUEID')/
CALL owner.SW_GRAFTCOUNT ('CUSTREQ', -1, -1, 101, 'GRAFT01', 'UNIQUEID', 1)/
```

---

## 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>       | varchar(8),  |
| <i>proc_maj_ver</i>    | integer,     |
| <i>proc_min_ver</i>    | integer,     |
| <i>case_number</i>     | numeric(20), |
| <i>graft_step_name</i> | varchar(8),  |
| <i>graft_id</i>        | varchar(49), |
| <i>graft_count</i>     | integer)     |

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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).



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 (
    proc_name          varchar(8),
    proc_maj_ver       integer,
    proc_min_ver       integer,
    jump_step          varchar(8),
    case_number        numeric(20),
    jump_reason        varchar(24),
    user_id            varchar(24))
```

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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.

**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).



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”.

---

```
CALL owner.SW_JUMPTO ('CUSTREQ', -1, -1, 'STEP5', 102, 'Administrator-initiated  
Jump', 'swadmin')/
```

---

## 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 (
    tgt_proc_name          varchar(8),
    tgt_proc_maj_ver       integer,
    tgt_proc_min_ver       integer,
    src_step_name          varchar(8),
    tgt_step_name          varchar(1024),
    case_number            numeric(20),
    jump_reason            varchar(24),
    user_id               varchar(24))
```

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 below.
- *tgt\_proc\_min\_ver* is either the minor version number of the *tgt\_proc\_name* procedure, or -1. See the notes below.
- *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).



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”.

---

```
CALL owner.SW_JUMPTO_MULTI ('CARPOOL', -1, -1, 'REQUEST', 'ALLOCATE,REFUSED', 110,
'Request Refused', 'swadmin')/
```

---

## 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 (
    proc_name          varchar(8),
    proc_maj_ver       integer,
    proc_min_ver       integer,
    case_number        numeric(20))
```

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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).



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.

---

```
CALL owner.SW_PURGE ('CUSTREQ', -1, -1, 103)/
```

---

# 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 (  
    proc_name          varchar(8) ,  
    proc_maj_ver       integer ,  
    proc_min_ver       integer ,  
    case_number        numeric(20))
```

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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).



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>    | varchar(8),  |
| <i>proc_maj_ver</i> | integer,     |
| <i>proc_min_ver</i> | integer,     |
| <i>case_number</i>  | numeric(20), |
| <i>user_id</i>      | varchar(24), |
| <i>suspend_type</i> | integer)     |

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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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 should always be 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).



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.

---

```
CALL owner.SW_SUSPEND ('CUSTREQ', -1, -1, 103, 'swadmin', 2)/
```

---

# 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 below.
- *proc\_min\_ver* is either the minor version number of the *proc\_name* procedure, or -1. See the notes below.
- *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).



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.

---

```
CALL owner.SW_ACTIVATE ('CUSTREQ', -1, -1, 103, 'swadmin')/
```

---

## 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\_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 if caching is enabled, and is controlled by the local variable `cache_size` in the `SW_GET_SEQUENCE` stored 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 restores using the default Mbox set for all operations.

**Example**            The following example shows how to set another Mbox set BGMGBSETB for bulk operations. Remember to create the sw\_db\_bgqueue\_3 and sw\_db\_bgqueue\_4 physical queues first. For more information about queue processing and Mbox set creation, see [Processing Queues on page 244](#).

---

```
# Step 1. Add two new message queues.
#
swadm add_queue BGMBOX3 Local 0003:swpro.sw_db_bgqueue_3
swadm add_queue BGMBOX4 Local 0003:swpro.sw_db_bgqueue_4

# Step 2. Add a new Mbox set.
#
swadm add_mboxset BGMGBSETB Local

# Step 3. View Mbox and queue IDs.
#
swadm show_mboxsets v
swadm show_queues

# Step 4. Add the BGMBOX3 and BGMBOX4 message queues to the BGMGBSETB Mbox set (
6 is the Mboxset ID of the BGMGBSETB Mbox set,
8 is the queue ID of the BGMBOX3 message queue, and
9 is the queue ID of the BGMBOX4 message queue.)
#
swadm add_queue_to_mboxset 6 8
swadm add_queue_to_mboxset 6 9

# Step 5. Set the BGMGBSETB Mbox set for bulk case starts.
#
CALL swpro.SW_SET_MBOX(6)/
```

```
# Step 6. Start the bulk cases.  
#  
CALL swpro.SW_ADD_PACK_DATA ('CustName', 'Allsop, J.A')/  
CALL swpro.SW_ADD_PACK_DATA ('CustID', '478163')/  
CALL swpro.SW_CASESTART ('CUSTREQ', -1, -1, 'Refund request', 'user35', '', '?', ?)/  
  
# Step 7. Restore using the default Mbox set.  
#  
CALL 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*    smallint)

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 on page 245](#) 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.

---

```
CALL swpro.SW_SET_PRIORITY (70)/
CALL swpro.SW_CASESTART ('CUSTREQ', -1, -1, 'Case1', 'user35', '', ?, ?)/
CALL swpro.SW_CASESTART ('CUSTREQ', -1, -1, 'Case2', 'user35', '', ?, ?)/
CALL swpro.SW_SET_PRIORITY (100)/
CALL swpro.SW_CASESTART ('CUSTREQ', -1, -1, 'Case3', 'user35', '', ?, ?)/
CALL swpro.SW_CASESTART ('CUSTREQ', -1, -1, 'Case4', 'user35', '', ?, ?)/
CALL swpro.SW_UNSET_PRIORITY ()/
CALL swpro.SW_CASESTART ('CUSTREQ', -1, -1, 'Case5', 'user35', '', ?, ?)/
```

---

## 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 on page 244](#) 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 on page 245](#) 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.

## 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 which is available in the current session:

```
TEMPORARY TABLE SSOLITE_DEBUG_DATA (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.

# SW\_SET\_DEBUG

---

The SW\_SET\_DEBUG procedure turns debugging on or off.

**Syntax**            SW\_SET\_DEBUG(  
                              *enable*                    smallint)

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.

## Appendix C    **Database Stored Procedures**

This appendix describes the package of database stored procedures.

### Topics

---

- [Overview, page 292](#)
- [CASENUM\\_FIND\\_GAPS, page 293](#)

## 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 [Chapter 4, Sequence Numbers, on page 43](#) 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 holds 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:



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.

---

```
call swpro.casenum_find_gaps(1, 26, 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](#)



## Appendix D Unused Tables

The following tables are created by the database creation script (`init2Kdb2.sql`), but are not currently used by the iProcess Engine:

- `pack_attach`
- `process_invqueue`
- `prounqid`
- `attachment`



Do not delete these tables. They are reserved for possible future use.

