

TIBCO iProcess[®] Engine (Oracle)

Administrator's Guide

*Software Release 11.4.1
April 2014*

Two-Second Advantage[®]



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

| | |
|---|-----------|
| 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 |
| Chapter 1 The TIBCO iProcess Engine Node | 1 |
| Table Relationships | 2 |
| nodes | 3 |
| Chapter 2 Process Sentinels. | 5 |
| Table Relationships | 6 |
| node_cluster | 7 |
| process_config | 9 |
| process_event_log | 12 |
| process_attributes | 14 |
| running_processes | 17 |
| active_logins | 19 |
| checksums | 21 |
| Chapter 3 Mbox Sets and Message Queues | 23 |
| Table Relationships | 24 |
| iql_queues | 25 |
| mbox_set | 28 |
| mbox_set_group | 30 |
| Oracle AQ Queue Tables and Queues. | 32 |
| Chapter 4 Sequence Numbers | 37 |
| About Sequence Numbers. | 38 |
| Table Relationships | 40 |

| | |
|---|-----------|
| sequences | 41 |
| Chapter 5 Procedures | 43 |
| Table Relationships | 44 |
| proc_index | 45 |
| iap_monitor | 49 |
| iap_field | 51 |
| iap_activity | 53 |
| iap_global | 55 |
| proc_version | 57 |
| procedure_lock | 60 |
| proc_instance | 62 |
| proc_audit | 65 |
| proc_defn | 68 |
| proc_deadline | 72 |
| proc_event | 75 |
| wqd_delta_subscriptions | 78 |
| Chapter 6 Procedure Management | 79 |
| About Procedure Objects | 80 |
| Table Relationships | 81 |
| pm_objects | 82 |
| pm_objects_lock | 85 |
| pmobjects_security | 88 |
| proc_mgt_hierarchy | 90 |
| Chapter 7 Cases | 93 |
| Table Relationships | 94 |
| case_information | 95 |
| outstanding_addr | 99 |
| wait | 102 |
| wait_step | 104 |
| status | 106 |
| case_data | 108 |
| audit_trail | 111 |
| memo | 114 |

| | |
|---|------------|
| predict | 117 |
| predict_lock | 121 |
| case_deadline_event | 123 |
| case_event | 126 |
| casenum_gaps | 130 |
| Chapter 8 Work Items | 133 |
| Table Relationships | 134 |
| staffo | 135 |
| pack_data | 139 |
| pack_memo | 141 |
| qaccess | 144 |
| Chapter 9 Case Data Queue Parameters | 147 |
| Table Relationships | 148 |
| cdqp_def | 149 |
| cdqp_cfg | 151 |
| Chapter 10 Queue Participation and Redirection | 153 |
| Table Relationships | 154 |
| part_defn | 155 |
| part_list | 157 |
| redir_defn | 159 |
| Chapter 11 Administrative Tables | 161 |
| Table Relationships | 162 |
| flag_table | 163 |
| version | 166 |
| Chapter 12 Users and Work Queues | 169 |
| About User Tables | 170 |
| Table Relationships | 171 |
| user_names | 172 |
| user_attrib | 174 |
| user_settings | 177 |
| user_values | 178 |
| user_memb | 181 |

| | |
|---|------------|
| leavers | 183 |
| tsys_user_names | 185 |
| tsys_user_attrib | 186 |
| tsys_user_values | 187 |
| tsys_user_memb | 188 |
| Chapter 13 Roles | 189 |
| About Roles | 190 |
| Table Relationships | 191 |
| role_users | 192 |
| tsys_role_users | 194 |
| Chapter 14 TIBCO iProcess Tables | 195 |
| About TIBCO iProcess Tables | 196 |
| Table Relationships | 197 |
| db_s_names | 198 |
| db_s_fields | 200 |
| db_s_values | 202 |
| tsys_db_s_names | 204 |
| tsys_db_s_fields | 205 |
| tsys_db_s_values | 206 |
| str_db_s_names | 207 |
| str_db_s_fields | 208 |
| ttmp_db_s_names | 209 |
| ttmp_db_s_fields | 210 |
| ttmp_db_s_values | 211 |
| Chapter 15 Lists | 213 |
| About Lists | 214 |
| Table Relationships | 215 |
| list_names | 216 |
| list_values | 218 |
| tsys_list_names | 220 |
| tsys_list_values | 221 |
| ttmp_list_names | 222 |
| ttmp_list_values | 223 |

| | |
|--|------------|
| Chapter 16 iProcess Server Plug-ins | 225 |
| Table Relationships | 226 |
| eai_registry | 227 |
| Chapter 17 Firewall Port Ranges | 229 |
| Table Relationships | 230 |
| port_range | 231 |
| port_range_active | 234 |
| port_range_conf | 236 |
| port_range_nodes | 238 |
| Chapter 18 WQS/WIS Shared Memory | 241 |
| Table Relationships | 242 |
| wqs_index | 243 |
| Appendix A Views | 247 |
| Appendix B STORAGE Configuration Macro Values | 249 |
| TABLESIZE | 250 |
| TABLEPCTINCREASE | 251 |
| INDEXSIZE | 252 |
| TABLESPACE | 253 |
| INDEXSPACE | 254 |
| AQTABLESPACE | 255 |
| Appendix C SSOLite Stored Procedures | 257 |
| Overview | 258 |
| Using SSOLite Stored Procedures | 259 |
| Processing Asynchronous Message | 259 |
| Transactional Processing | 259 |
| Handling Exceptions | 259 |
| Processing Queues | 261 |
| Prioritizing Messages | 262 |
| Data Procedures | 264 |
| Command Procedures | 271 |
| Control Procedures | 294 |
| Debug Procedures | 304 |

| | |
|--|------------|
| Appendix D Database Stored Procedures | 309 |
| Overview | 310 |
| CASENUM_FIND_GAPS | 311 |
| Appendix E Unused Tables | 315 |

Preface

This guide describes the TIBCO iProcess Engine (Oracle) 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"> on a Windows server (on the C: drive) <p>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.</p> on a UNIX or Linux server <p>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 <code>\$SWDIR/bin/swutil</code> directory.</p> <p>Note: On a UNIX or Linux system, the environment variable <code>\$SWDIR</code> 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 foo process.</p> |

Table 1 General Typographical Conventions (Cont'd)

| Convention | Use |
|---|--|
| bold code font | <p>Bold code font is used in the following ways:</p> <ul style="list-style-type: none"> • In procedures, to indicate what a user types. For example: Type admin. • In large code samples, to indicate the parts of the sample that are of particular interest. • In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, MyCommand is enabled: MyCommand [enable disable] |
| <i>italic font</i> | <p>Italic font is used in the following ways:</p> <ul style="list-style-type: none"> • To indicate a document title. For example: See <i>TIBCO ActiveMatrix BusinessWorks Concepts</i>. • To introduce new terms. For example: A portal page may contain several portlets. <i>Portlets</i> are mini-applications that run in a portal. • To indicate a variable in a command or code syntax that you must replace. For example: MyCommand <i>PathName</i> |
| 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> <pre>MyCommand param1 param2 param3</pre> |
| { } | <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> <pre>MyCommand {param1 param2} {param3 param4}</pre> <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> <pre>MyCommand {param1 param2} {param3 param4}</pre> <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> <pre>MyCommand param1 [param2] {param3 param4}</pre> |

Connecting with TIBCO Resources

How to Join TIBCOmmunity

TIBCOmmunity 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. TIBCOmmunity 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.

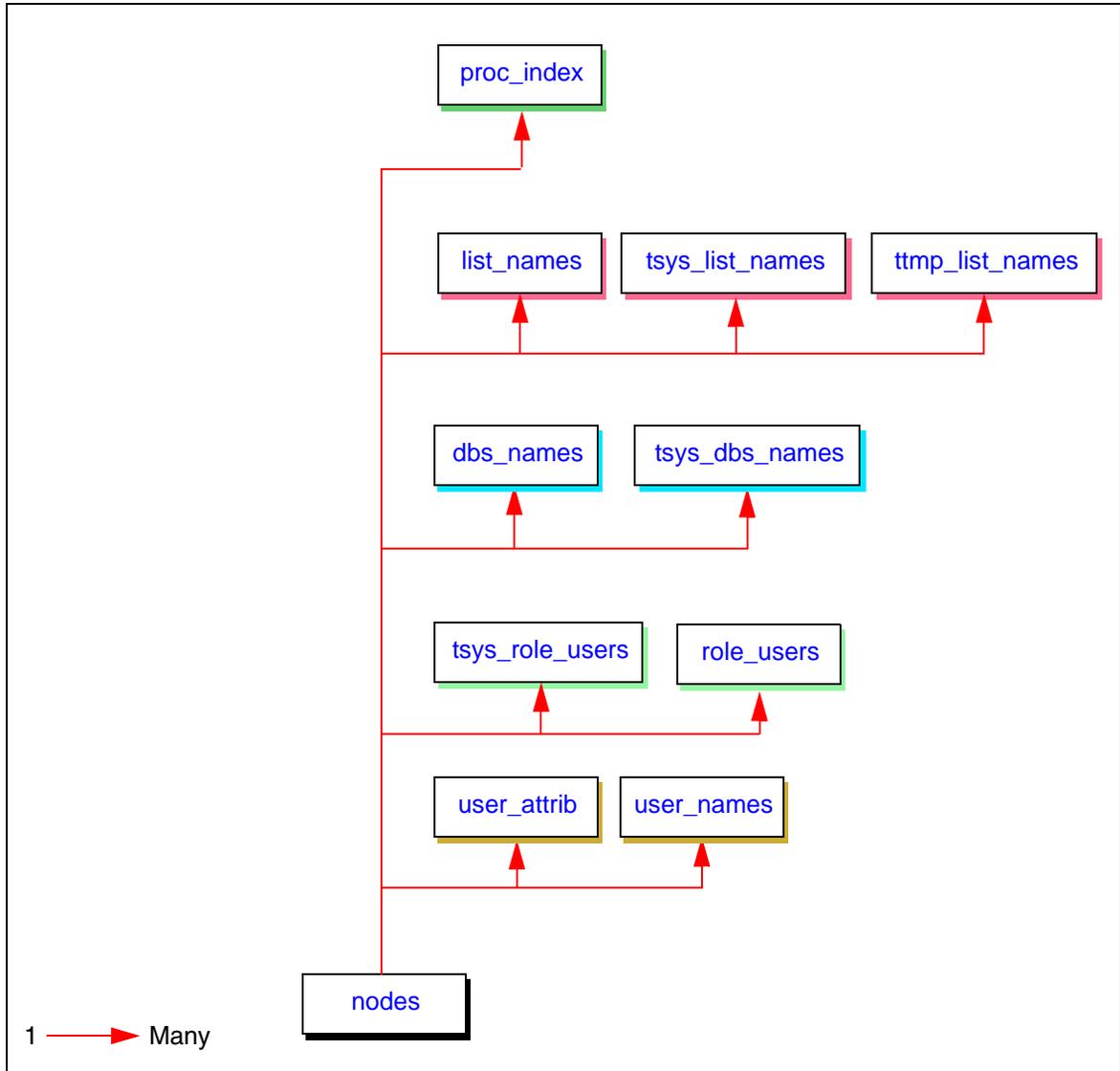
Topics

- [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          number(5)          NOT NULL,
  node_name        varchar2(24)       NOT NULL,
  dir_name         varchar2(28)       NOT NULL,
  mail_addr        varchar2(149)      NULL,
  mail_cert        varchar2(31)       NULL,
  mail_type        number(5)          NOT NULL,
  node_public      number(1)          NOT NULL,
  node_slave       number(1)          NOT NULL,
  node_deleted     number(1)          NOT NULL,
  rpc_majvers      number(3)          NOT NULL,
  rpc_minvers      number(3)          NOT NULL,
  server_majvers   number(3)          NOT NULL,
  server_minvers   number(3)          NOT NULL)
```

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

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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.

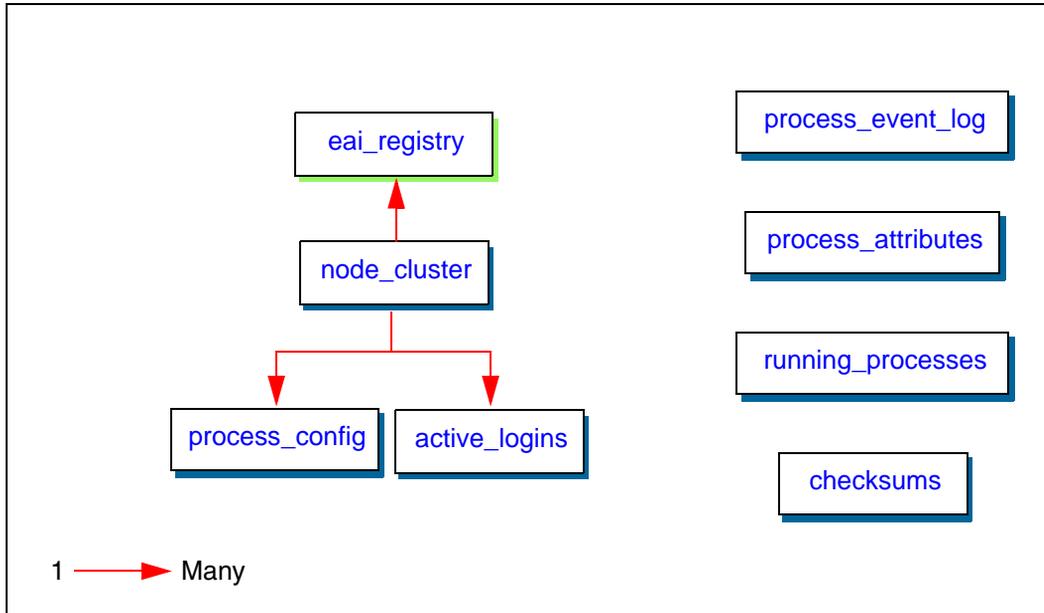
Topics

- [Table Relationships](#), page 6
- [node_cluster](#), page 7
- [process_config](#), page 9
- [process_event_log](#), page 12
- [process_attributes](#), page 14
- [running_processes](#), page 17
- [active_logins](#), page 19
- [checksums](#), 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    number(5)           NOT NULL,
  physical_machine_name varchar2(256)       NOT NULL,
  master                number(1)         NOT NULL,
  check_error_files    number(1)         NOT NULL,
  machine_comment      varchar2(256)     NULL)
```

| Column | Description |
|------------------------------------|--|
| <code>logical_machine_id</code> | Unique ID for this server. |
| <code>physical_machine_name</code> | If a UNIX server, the name of this server (as returned by the UNIX <code>uname</code> command). If a Windows server, then the name of this server or the Microsoft Windows cluster network name. |
| <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) | Index Tablespace |
|------------------------------|---------------------------------|-----------------------------|
| <code>pk_node_cluster</code> | <code>logical_machine_id</code> | <code>TINYINDEXSPACE</code> |

Unique Key The following unique key is defined for this table.

| Key Name | Column(s) | Index Tablespace |
|-------------------------------|------------------------------------|-----------------------------|
| <code>unq_node_cluster</code> | <code>physical_machine_name</code> | <code>TINYINDEXSPACE</code> |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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

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

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

process_config

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

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

Structure The `process_config` table has the following structure:

```
TABLE process_config(
  logical_machine_id    number(5)           NOT NULL,
  logical_process_name  varchar2(10)        NOT NULL,
  logical_process_instance number(5)       NOT NULL,
  enabled               number(1)          NOT NULL,
  persistent            number(1)          NOT NULL,
  last_known_status    varchar2(20)       NOT NULL,
  status_comment        varchar2(255)     NULL)
```

| Column | Description |
|---------------------------------------|---|
| <code>logical_machine_id</code> | ID of the server where this process instance runs, as defined in the node_cluster table. |
| <code>logical_process_name</code> | Logical name of this process instance. Note: See "Administering iProcess Engine Server Processes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of logical process names. |
| <code>logical_process_instance</code> | 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. Note: Any row in which the <code>persistent</code> 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. Either: STARTING, RUNNING, PAUSED, SUSPENDED, SHUTTING DOWN or STOPPED. Note: The process_event_log 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) | Index Tablespace |
|-------------------|--|------------------|
| pk_process_config | logical_machine_id logical_process_name logical_process_instance | SMALLINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|-------------------|--------------------|------------------------|
| fk_process_config | logical_machine_id | node_cluster |

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-----------------------|--------------------|-----------------|
| idx_process_config_fk | logical_machine_id | SMALLINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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

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

| A row is... | When... |
|-------------|--|
| added | a new process instance is added, either at installation or by using the <i>SWDIR\util\swadm</i> , <i>SWDIR\util\swsvrmgr</i> utilities or the iProcess Server Manager. |
| updated | a process instance's settings or status are updated, either by system activity, or by using the <i>SWDIR\util\swadm</i> , <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  number(5)           NOT NULL,
  logical_process_name varchar2(10)       NOT NULL,
  logical_process_instancenumber(5)       NOT NULL,
  process_id          number(10)          NOT NULL,
  process_status      number(5)           NOT NULL,
  process_status_commentvarchar2(255)    NULL,
  timestamp           date                NOT NULL)
```

| Column | Description |
|--------------------------|--|
| logical_machine_id | ID of the server where the process instance that this event applies to is running, as defined in the node_cluster table. |
| logical_process_name | Logical name of the process that this event applies to. Note: See "Administering iProcess Engine Server Prozesse" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of logical process names. |
| logical_process_instance | ID of the process instance that this event applies to, as defined in the process_config table. |
| process_id | Process ID (PID) of the process instance that this event applies to. |
| process_status | Status change event that occurred for the specified process instance. One of the following: <ul style="list-style-type: none"> • 3000 - process instance started. • 3001 - process instance stopping. • 3002 - process instance stopped. • 3003 - process instance died. • 3004 - process instance paused. • 3005 - process instance unpaused. |
| process_status_comment | Description of the process_status entry, as reported to the Process Sentinels by the process. |
| timestamp | Date and time that this event occurred. |

| | |
|---------------------|--|
| Primary Key | None. |
| Foreign Keys | None. |
| Indexes | None. |
| Storage | The following STORAGE values are defined for this table. |

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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

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

| A row is... | When... |
|-------------|--|
| added | a process starts, receives a shutdown command, or shuts down. |
| updated | never. |
| deleted | never. Note: Because rows are never deleted automatically, TIBCO recommend that you regularly monitor the size of this table and delete or archive rows manually if you need to. |

process_attributes

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

Structure The `process_attributes` table is structured as follows:

```
TABLE process_attributes (
  logical_machine_id  number(5)           NOT NULL,
  logical_process_name varchar2(10)       NOT NULL,
  logical_process_instance number(5)     NOT NULL,
  attribute_name      varchar2(50)       NOT NULL,
  attribute_value     varchar2(1024)    NOT NULL,
  attribute_type      varchar2(2)       NOT NULL)
```

| Column | Description |
|---------------------------------------|---|
| <code>logical_machine_id</code> | ID of the server where the process instance that this attribute applies to is running, as defined in the node_cluster table. A value of 0 means that this attribute applies to all servers that are part of this node. |
| <code>logical_process_name</code> | Logical name of the process that this attribute applies to. A value of ALL means that this attribute applies to all processes on the indicated server. Note: See "Administering iProcess Engine Server Processes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of logical process names. |
| <code>logical_process_instance</code> | ID of the process instance that this attribute applies to, as defined in the process_config table. A value of 0 means that this attribute applies to all instances of the indicated process. |
| <code>attribute_name</code> | Name of this process attribute. Note: See "Administering Process Attributes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of the available process attributes. |
| <code>attribute_value</code> | Value of this process attribute. Note: See "Administering Process Attributes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of the valid values for each process attributes. |

| Column | Description |
|----------------|---|
| attribute_type | Type of this process attribute: either I (Integer), C (Character) or S (String). Note: All attribute_values are stored as strings in this table. This value determines how the value is returned to the <i>SWDIR\bin\swadm</i> interface. |

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

| Key Name | Column(s) | Index Tablespace |
|-----------------------|--|------------------|
| pk_process_attributes | logical_machine_id logical_process_name logical_process_instance attribute_name | SMALLINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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  number(5)           NOT NULL,
  logical_process_name varchar2(10)       NOT NULL,
  logical_process_instance number(5)     NOT NULL,
  process_id          number(10)        NOT NULL,
  port_number         number(5)         NOT NULL)
```

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

Primary Key None.

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---------------------------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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    number(5)           NOT NULL,
  logical_process_name  varchar2(10)        NOT NULL,
  logical_process_insta number(5)           NOT NULL,
  user_name             varchar2(64)       NOT NULL,
  user_id              varchar2(37)       NOT NULL,
  process_id           number(10)         NOT NULL,
  filsh               number(5)           NOT NULL,
  windows              number(2)          NOT NULL,
  station_id          varchar2(32)        NOT NULL)
```

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

1. This key enforces the DELETE CASCADE referential action.

Indexes The following indexes are defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|----------------------|--------------------|------------------|
| idx_active_logins_fk | logical_machine_id | SMALLINDEXSPACE |
| idx_active_logins | user_id | MEDIUMINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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

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

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

checksums

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

Structure The checksums table has the following structure:

```
TABLE checksums (
  area_id          number(5)          NOT NULL,
  area_name        varchar2(20)       NOT NULL,
  check_sum        varchar2(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) | Index Tablespace |
|--------------|-----------|------------------|
| pk_checksums | area_id | TINYINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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

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

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

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 [Oracle AQ Queue Tables and Queues](#) which provide the underlying message queueing system used by the iProcess message queues.

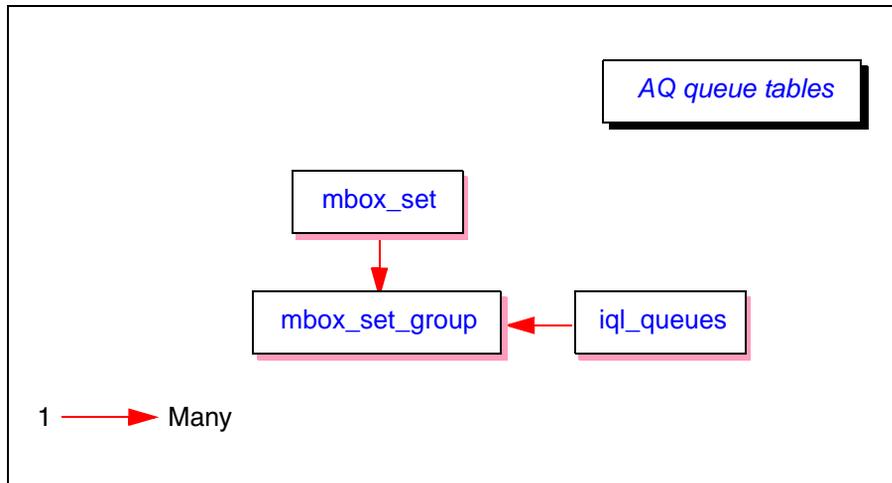
Topics

- [Table Relationships, page 24](#)
- [iql_queues, page 25](#)
- [mbox_set, page 28](#)
- [mbox_set_group, page 30](#)
- [Oracle AQ Queue Tables and Queues, page 32](#)

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          number(5)          NOT NULL,
  queue_name        varchar2(24)       NOT NULL,
  queue_type        number(2)          NOT NULL,
  queue_phys_descr  varchar2(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 Oracle AQ queue that is used to hold this message queue. See: Format of the Oracle AQ QueueID on page 25 for a description of the format used for this value. Oracle AQ Queue Tables and Queues on page 32 for more information about the required structure of AQ queue tables and queues, and how to create them. |

Format of the Oracle AQ QueueID

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

```
0001:[instance]:queue_table:queue
```

where:

- 0001 indicates that the remainder of the string uses Oracle AQ format.
- *instance* is the Oracle Service Name of the Oracle instance that holds the AQ queue. If this is blank, the default local instance is assumed.
- *queue_table* is the name of the AQ queue table that holds the AQ queue.
- *queue* is the name of the AQ queue used to hold this message queue.

For example, the entry:

0001::bgmboxtable1:bgmboxqueue1

describes the AQ queue called `bgmboxqueue1` contained in queue table `bgmboxtable1` on the default Oracle instance.

Default Message Queues and AQ Tables

When the iProcess Engine is installed, the `init2Kora.sql` (on UNIX, or on Windows `init2Kora_tok.sql`) script creates the following default set of message queues and AQ queue tables required by the system.

| Queue Name | Oracle AQ Queue ID |
|--------------|---|
| BGMBBOX1 | 0001::bgmboxtable1:bgmboxqueue1 |
| BGMBBOX2 | 0001::bgmboxtable2:bgmboxqueue2 |
| WISMBOX1 | 0001::wismboxtable1:wismboxqueue1 |
| WISMBOX2 | 0001::wismboxtable2:wismboxqueue2 |
| PREDICTMBOX1 | 0001::predictmboxtable1:predictmboxqueue1 |
| PREDICTMBOX2 | 0001::predictmboxtable2:predictmboxqueue2 |

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

| Key Name | Column(s) | Index Tablespace |
|---------------|-----------|------------------|
| pk_iql_queues | queue_id | TINYINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

Table Activity The `iq1_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 <code>SWDIR\util\swadm</code> utility. |
| updated | a message queue's details are updated, using the <code>SWDIR\util\swadm</code> utility. |
| deleted | a message queue is deleted from the node, using the <code>SWDIR\util\swadm</code> 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          number(5)          NOT NULL,
  mbox_set_name        varchar2(32)       NOT NULL,
  mbox_set_msgtype     number(2)         NOT NULL)
```

Each row provides the following information about a Mbox set.

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

Default Mbox Sets and Message Queues

When the iProcess Engine is installed, the `init2Kora.sql` (on UNIX, or on Windows `init2Kora_tok.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 |
| WDMBSET | WISMBBOX1, WISMBBOX2 |
| PREDICTMBSET | PREDICTMBOX1, PREDICTMBOX2 |

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

| Key Name | Column(s) | Index Tablespace |
|--------------------------|--------------------------|-----------------------------|
| <code>pk_mbox_set</code> | <code>mbox_set_id</code> | <code>TINYINDEXSPACE</code> |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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          number(5)          NOT NULL,
  mbox_queue_id        number(5)          NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|--------------------------------|--|--------------------------------|
| <code>pk_mbox_set_group</code> | <code>mbox_set_id</code> <code>mbox_queue_id</code> | TINYINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|---|--------------------------|----------------------------|
| <code>fk_mbox_set_id¹</code> | <code>mbox_set_id</code> | mbox_set |
| <code>fk_mbox_queue_id</code> | <code>queue_id</code> | iql_queues |

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

Indexes The following indexes are defined for this table.

| Index Name | Indexed Column(s) | Tablespace |
|-----------------------------------|----------------------------|--------------------------------|
| <code>idx_mbox_set_id_fk</code> | <code>mbox_set_id</code> | TINYINDEXSPACE |
| <code>idx_mbox_queue_id_fk</code> | <code>mbox_queue_id</code> | TINYINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

Table Activity The `mbx_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. |

Oracle AQ Queue Tables and Queues

Oracle Advanced Queuing (AQ) provides the underlying message queuing system used by the iProcess message queues. Each message queue defined in the [iql_queues](#) table must be mapped to an AQ queue, which in turn must be stored in an AQ table.

When the iProcess Engine is installed, the `init2Kora.sql` script (on UNIX, or on Windows `init2Kora_tok.sql`) creates the default set of AQ tables, queues and object types required by the system (see [Default Message Queues and AQ Tables on page 26](#).)



The `init2Kora.sql` script (on UNIX, or on Windows `init2Kora_tok.sql`) also creates a non-persistent Oracle AQ queue called `sw_system_events`. This queue is used to route all iProcess events to the iProcess event manager. (iProcess event management is provided by the Oracle Event Manager on UNIX, and by the iProcess Events COM+ application on Windows.)

If you subsequently decide to add additional message queues, you must use the Oracle DBMS_AQADM package to manually create the Oracle AQ queue tables and queues needed by those message queues. The following sections explain how to do this.



For a detailed description of the DBMS_AQADM package procedures referred to in this section, see the *Oracle9i Supplied PL/SQL Packages and Types Reference* guide in your Oracle documentation set.

SWLOCALMSG Object Type

An AQ queue table must be associated with one particular message type, which must already exist when you create the AQ queue table.

All AQ queue tables used by the iProcess Engine must use the SWLOCALMSG object type, which is created when the iProcess Engine is installed.

```
CREATE TYPE TABLE_OWNER.SWLOCALMSG AS OBJECT (
    msginfo      VARCHAR2(500),
    node         varchar2(24),
    contenttype  number(2),
    data         VARCHAR2(1000)
);
```

AQ Queue Tables

Use the `DBMS_AQADM.CREATE_QUEUE_TABLE` procedure to create an AQ queue table. The following parameters are required.

| Column | Description |
|---------------------------|--|
| <i>queue_table</i> | Unique name of this queue table. This name is used in the <code>queue_phys_descr</code> column of the <code>iql_queues</code> table. |
| <i>queue_payload_type</i> | Object type of data stored by this queue table. This value must be <code>swlocalmsg</code> . |
| <i>storage_clause</i> | Tablespace to be used by this queue table. Note: Each queue table can be associated with a different tablespace, which in turn can be associated with a different datafile. Distributing queue tables in this way provides an easy way of reducing I/O contention and optimizing performance. See Appendix B on page 249 for more information. |
| <i>sort_list</i> | Sort criteria by which messages are sorted in ascending order and dequeued based on this order. |
| <i>comment</i> | Description of this queue table. |
| <i>compatible</i> | Lowest database version that this queue table is compatible with. This value must be <code>8.1</code> . |

The following example, from the `init2Kora.sql` script, shows the SQL used to create one of the default background Mbox tables that is created when the iProcess Engine is installed.

```
DBMS_AQADM.CREATE_QUEUE_TABLE (
  queue_table => 'TABLE_OWNER.bgmboxtable1',
  queue_payload_type => 'TABLE_OWNER.swlocalmsg',
  sort_list => ' priority, enq_time',
  storage_clause => 'tablespace AQTABLESPACE',
  comment => 'default background mbox',
  compatible => '8.1');
```

AQ Queues

To create an AQ queue and make it available to be used as an iProcess message queue:

1. Use the `DBMS_AQADM.CREATE_QUEUE` procedure to create the AQ queue. The following parameters are required.

| Column | Description |
|--------------------|---|
| <i>queue_name</i> | Unique name of this queue. This name is used in the <code>queue_phys_descr</code> column of the <code>iq1_queues</code> table. |
| <i>queue_table</i> | Name of the queue table that holds this queue. This name is used in the <code>queue_phys_descr</code> column of the <code>iq1_queues</code> table. Note: Multiple queues can be stored in the same queue table. |
| <i>retry_delay</i> | Delay time, in seconds, before a message is scheduled for processing again after an application rollback. |
| <i>max_retries</i> | Number of times that a dequeue with the REMOVE mode can be attempted on a message. The count is incremented when the application issues a rollback after executing the dequeue. The message is moved to the default exception queue when it reaches its <code>max_retries</code> . |

The following example, from the `init2Kora.sql` script, shows the SQL used to create one of the default background Mbox queues that is created when the iProcess Engine is installed.

```
DBMS_AQADM.CREATE_QUEUE (queue_name => 'TABLE_OWNER.bgmbboxqueue1', queue_table =>
'TABLE_OWNER.bgmbboxtable1', retry_delay => 300, max_retries => 12);
```

2. Use the `DBMS_AQADM.START_QUEUE` procedure to start the queue.
3. Use the `DBMS_AQADM.GRANT_QUEUE_PRIVILEGE` procedure to grant the privileges on the queue. The following parameters are required.

| Column | Description |
|------------------|---|
| <i>privilege</i> | The AQ queue privilege to grant. This value must be ALL (which means both ENQUEUE and DEQUEUE). |

| Column | Description |
|---------------------|--|
| <i>queue_name</i> | Name of the queue. |
| <i>grantee</i> | Grantee. This value must be the name of the iProcess Oracle Foreground user. Note: The Oracle Foreground user is defined in the third entry (after the second backslash character) on line 9 of the <i>SWDIR\etc\staffpms</i> file. For example: 3\swpro\swuser\swpro\staffdb1\10\0 |
| <i>grant_option</i> | Defines whether the access privilege is granted with the GRANT option or not. This value must be false. |

The following example, from the `init2Kora.sql` script, shows the SQL used to grant the required privileges on one of the default background Mbox queues that is created when the iProcess Engine is installed.

```
dbms_aqadm.grant_queue_privilege (privilege => 'all', queue_name =>
'TABLE_OWNER.bgmboxqueue1', grantee => 'FOREGROUND_USER', grant_option => false )
```

Example Suppose that the volume of messages handled by your system has increased significantly, and the default messages queues are no longer able to cope. To deal with the additional load you have decided that you need to add a new BGMBOX3 message queue to the BGMBSET Mboxset. This queue requires a new `bgmboxqueue3` Oracle AQ queue and `bgmboxtable3` queue table .



This example uses UNIX syntax.

To do this:

1. Use SQLPLUS to connect to the Oracle instance.

```
$ORACLE_HOME/bin/sqlplus
connect swpro/staffpro1;
```

2. Create an AQ queue table called `bgmboxtable3`.

```
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE (queue_table => 'swpro.bgmboxtable3',
queue_payload_type => 'swpro.swlocalmsg', comment => '3rd background mbox', compatible =>
8.1);
```

3. Create an AQ queue called `bgmboxqueue3` in this queue table.

```
EXECUTE DBMS_AQADM.CREATE_QUEUE (queue_name => 'swpro.bgmboxqueue3', queue_table =>
'swpro.bgmboxtable3', retry_delay => 300, max_retries => 12);
```

4. Start the `bgmboxqueue3` queue.

```
EXECUTE DBMS_AQADM.START_QUEUE (queue_name => 'swpro.bgmboxqueue3');
```

5. Grant the required privileges on the `bgmboxqueue3` queue.

```
EXECUTE dbms_aqadm.grant_queue_privilege (privilege => 'all', queue_name =>
'bgmboxqueue3', grantee => 'swuser', grant_option => false );
```

6. Commit the changes, disconnect and exit.

```
commit;
disconnect;
exit;
```

7. Use the `$SWDIR\util\swadm` utility to add a new message queue called `BGMBX3`, which uses the `bgmboxqueue3` queue.

```
cd $SWDIR/util
swadm ADD_QUEUE BGMBX3 Local 0001::bgmboxtable3:bgmboxqueue3
```

8. Add the `BGMBX3` 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 38](#)
- [Table Relationships, page 40](#)
- [sequences, page 41](#)

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... |
|-----------------------------|--|------------------------------------|
| <code>o_reqid</code> | <code>staffo</code> | Work item |
| <code>casenum</code> | <code>case_information</code> | Case |
| <code>proc_id</code> | <code>proc_index</code> | Procedure |
| <code>wait_id</code> | <code>wait</code> | Outstanding Wait |
| <code>def_id</code> | <code>cdqp_def</code> | CDQP definition |
| <code>cfg_id</code> | <code>cdqp_cfg</code> | CDQP value |
| <code>monitor_id</code> | <code>iap_monitor</code> | Procedure that IAP is monitoring |
| <code>provider_id</code> | <code>eaiws_jms_provider*</code> | JMS provider |
| <code>destination_id</code> | <code>eaiws_jms_destinati on*</code> | JMS endpoints for for JMS provider |

*. 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 (`init2Kora.sql` on UNIX, `init2Kora_tok.sql` on Windows).

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          number(5)          NOT NULL,
  seq_val         number(20)         NOT NULL,
  seq_name        varchar2(24)       NOT NULL)
```

| Column | Description |
|----------|---|
| seq_id | Sequence ID of the associated seq_val value. One of the following values: 1 (o_reqid) 2 (casenum) 3 (proc_id) 4 (wait_id) 5 (def_id) 6 (cfg_id) |
| seq_val | Current sequence number value for the sequence defined by seq_id. |
| seq_name | Name of the associated seq_id column. One of the following values: REQID CNUM PROC WAIT CDQP_DEF CDQP Note: This value is not currently used by the iProcess Suite. |

Primary Key None.

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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.

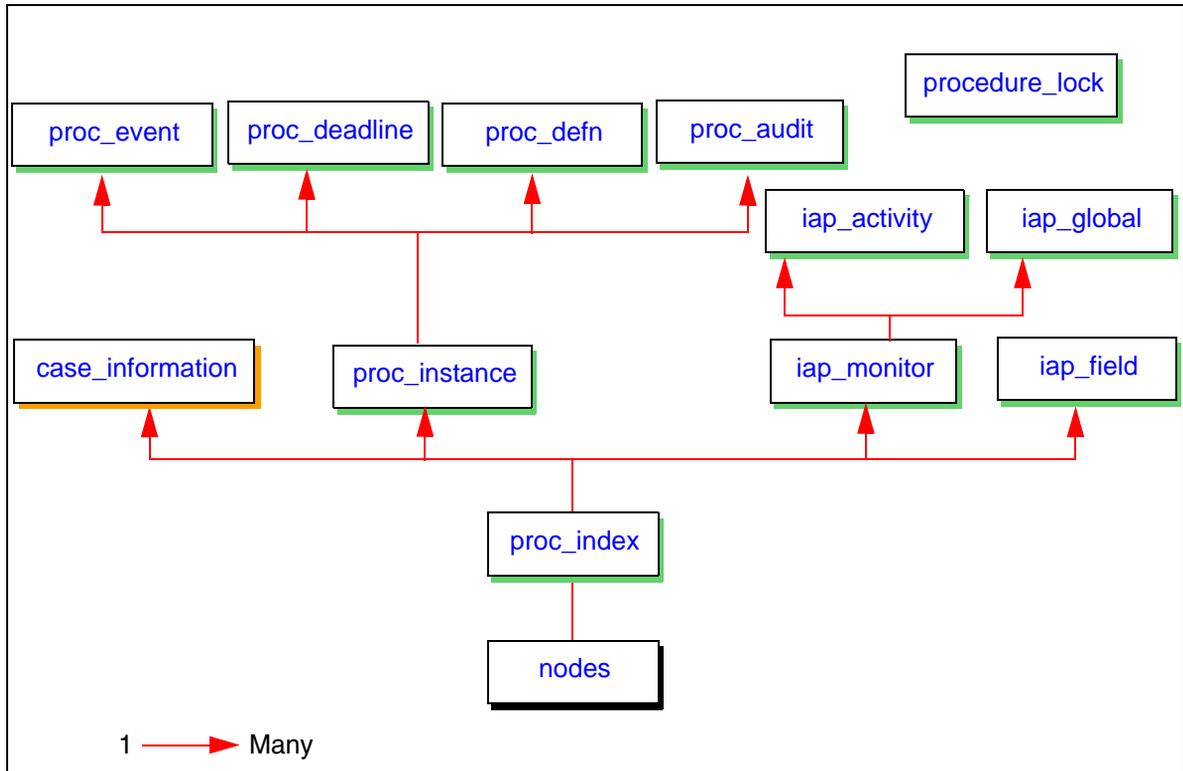
Topics

- [Table Relationships, page 44](#)
- [proc_index, page 45](#)
- [iap_monitor, page 49](#)
- [iap_field, page 51](#)
- [iap_activity, page 53](#)
- [iap_global, page 55](#)
- [proc_version, page 57](#)
- [procedure_lock, page 60](#)
- [proc_instance, page 62](#)
- [proc_audit, page 65](#)
- [proc_defn, page 68](#)
- [proc_deadline, page 72](#)
- [proc_event, page 75](#)
- [wqd_delta_subscriptions, page 78](#)

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  proc_used_count  number(5)          NOT NULL,
  proc_name        varchar2(8)        ,
  proc_desc        varchar2(24)       ,
  proc_owner       varchar2(24)       ,
  dir_name         varchar2(12)       ,
  proc_used        number(1)          NOT NULL,
  work_days        number(1)          NOT NULL,
  auto_purge       number(1)          NOT NULL,
  networked        number(1)          NOT NULL,
  cdesc_type       number(1)          NOT NULL,
  ignore_blanks   number(1)          NOT NULL,
  is_predict        number(1)          NOT NULL,
  normalise_data   number(1)          NOT NULL,
  delay_purge      number(1)          NOT NULL,
  delay_value      varchar(512)       NULL)
```

| Column | Description |
|------------------------------|---|
| <code>node_id</code> | ID of the node that this procedure is defined on, as defined in the nodes table. |
| <code>proc_id</code> | Unique ID of this procedure, generated from the sequences table. |
| <code>proc_used_count</code> | Not used. |
| <code>proc_name</code> | Name of this procedure. Note: 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 user_names 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"> 0 means that the field is treated as an error and the step is delivered to the undelivered queue. 1 means that the field is not treated as an error. |
| is_predict | Flag that defines whether (1) or not (0) case prediction is enabled for this procedure. |
| normalise_data | Flag that defines whether (1) or not (0) case data normalization is enabled for this procedure. |
| delay_purge | Flag that defines whether or not to delay the auto-purge operation. <ul style="list-style-type: none"> 0 means that the auto-purge operation is not delayed. 1 means that the auto-purge operation is delayed. |

| Column | Description |
|-------------|---|
| delay_value | <p>The value of the delay_value column is:</p> <ul style="list-style-type: none"> If the value of the delay_purge column is 0, then the value of the delay_value column is NULL. If the value of the delay_purge column is 1, then the value of the delay_value column is specified in one of the following ways: <ul style="list-style-type: none"> The period of the delay in days. Delayed date and time expressions: <i>date expression</i>^<i>time expression</i> |

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

| Key Name | Column(s) | Index Tablespace |
|---------------|--------------------|------------------|
| pk_proc_index | proc_id node_id | SMALLINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|----------------------------|-----------|------------------------|
| fk_proc_index ¹ | 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 | Tablespace |
|-------------------|-------------------|-----------------|
| idx_proc_index_fk | node_id | SMALLINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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

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

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

iap_monitor

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

Structure The `iap_monitor` table has the following structure:

```
TABLE iap_monitor(
  node_id          number(5)          NOT NULL,
  proc_id          number(5)          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 <code>nodes</code> table. |
| <code>proc_id</code> | Unique ID of this procedure, generated from the <code>sequences</code> 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) | Index Tablespace |
|-----------------------------|-------------------------|------------------------------|
| <code>pk_iap_monitor</code> | <code>monitor_id</code> | <code>SMALLINDEXSPACE</code> |

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

| Key Name | Column(s) | Referenced in Table... |
|--|--|---|
| <code>fk_iap_monitor</code> ¹ | <code>proc_id</code> <code>node_id</code> | <code>proc_index</code> <code>nodes</code> |

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

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|---------------------------------|----------------------|------------------------------|
| <code>idx_iap_monitor_fk</code> | <code>node_id</code> | <code>SMALLINDEXSPACE</code> |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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      number(10)      NOT NULL,
  field_name      varchar2(31)    NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|---------------------------|--|----------------------------------|
| <code>pk_iap_field</code> | <code>monitor_id</code> <code>field_name</code> | MEDIUMINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|--|-------------------------|-----------------------------|
| <code>fk_iap_field</code> ¹ | <code>monitor_id</code> | iap_monitor |

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

Indexes The following index is defined for this table.

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

Storage The following `STORAGE` values are defined for this table.

| Value | Definition |
|---------|---------------------------------|
| Initial | MEDIUMTABLESIZE |

| Value | Definition |
|---------------------|--|
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

Table Activity

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

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

iap_activity

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

Structure The `iap_activity` table has the following structure:

```
TABLE iap_activity(
  monitor_id          number(10)          NOT NULL,
  activity_id         number(3)           NOT NULL,
  step_name           varchar2(8)        NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|------------------------------|---|----------------------------------|
| <code>pk_iap_activity</code> | <code>monitor_id</code> <code>activity_id</code> <code>step_name</code> | MEDIUMINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|--|-------------------------|-----------------------------|
| <code>fk_iap_activity¹</code> | <code>monitor_id</code> | iap_monitor |

1. This key enforces the DELETE CASCADE referential action.

Indexes The following index is defined for this table.

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

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  field_name       varchar2(31)      NOT NULL)
```

| Column | Description |
|-------------------------|---|
| <code>node_id</code> | ID of the node that this procedure is stored on, as defined in the nodes table. |
| <code>proc_id</code> | Unique ID of this procedure, generated from the sequences 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) | Index Tablespace |
|----------------------------|---|---------------------------------|
| <code>pk_iap_global</code> | <code>proc_id</code> <code>node_id</code> <code>field_name</code> | SMALLINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|---|--|---|
| <code>fk_iap_global</code> ¹ | <code>proc_id</code> <code>node_id</code> | proc_index nodes |

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

Indexes The following index is defined for this table.

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

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  major_vers      number(5)          NOT NULL,
  minor_vers      number(5)          NOT NULL,
  pd_version      number(7)
  pv_status       number(2)
  pv_user         varchar2(49)
  pv_comment      varchar2(128)     NULL,
  pv_created      date
  pv_modified     date
  pv_released     date
  pv_withdrawn   date
  pv_is_subproc   number(1)         )
```

| Column | Description |
|-------------------------|--|
| <code>node_id</code> | ID of the node that this procedure is stored on, as defined in the nodes table. |
| <code>proc_id</code> | Procedure number of the procedure associated with this version, as defined in the proc_index 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 proc_instance 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 user_names 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) | Index Tablespace |
|-----------------|--|------------------|
| pk_proc_version | node_id proc_id major_vers minor_vers | BIGINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|------------------------------|--------------------|------------------------|
| fk_proc_version ¹ | node_id 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 | Tablespace |
|---------------------|--------------------|---------------|
| idx_proc_version_fk | node_id proc_id | BIGINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---------------------|
| Initial | BIGTABLESIZE |
| Percentage Increase | BIGTABLEPCTINCREASE |
| Tablespace | BIGTABLESPACE |

Table Activity The `proc_version` table contains one row for every version of every procedure (or sub-procedure or sub-procedure parameter template) defined on the system. Rows are added, updated and deleted in the following situations.

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

procedure_lock

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

Structure The `procedure_lock` table has the following structure:

```
TABLE procedure_lock (
  node_id          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  lock_state       number(1)          NOT NULL,
  lock_owner       varchar2(24)      ,
  lock_date        date              NOT NULL,
  lock_reason      number(1)          NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|--------------------------------|--|--------------------------------|
| <code>pk_procedure_lock</code> | <code>proc_id</code> <code>node_id</code> | TINYINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  major_vers      number(5)          NOT NULL,
  minor_vers      number(5)          NOT NULL,
  pd_version      number(7)          NOT NULL,
  pi_first_step   varchar2(8)        ,
  pi_rpa_start    number(1)          ,
  pi_rpa_admin    number(1)          ,
  pi_has_eis_objs number(1)          ,
  pi_has_subprocs number(1)          )
```

| Column | Description |
|----------------------------|---|
| <code>node_id</code> | ID of the node that this instance is stored on, as defined in the nodes table. |
| <code>proc_id</code> | Procedure number of the procedure associated with this instance, as defined in the proc_index table. |
| <code>major_vers</code> | Major version number of the version associated with this instance, as defined in the proc_version table. |
| <code>minor_vers</code> | Minor version number of the version associated with this instance, as defined in the proc_version table. |
| <code>pd_version</code> | Instance number of this procedure definition. |
| <code>pi_first_step</code> | Start step for this instance of the procedure. |
| <code>pi_rpa_start</code> | Flag that defines whether (1) or not (0) Remote Procedure Access (RPA) case start restrictions are set in the procedure definition. |
| <code>pi_rpa_admin</code> | 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) | Index Tablespace |
|------------------|----------------------------------|------------------|
| pk_proc_instance | node_id proc_id pd_version | BIGINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|-------------------------------|--------------------|------------------------|
| fk_proc_instance ¹ | node_id 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 | Tablespace |
|----------------------|--------------------|---------------|
| idx_proc_instance_fk | node_id proc_id | BIGINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---------------------|
| Initial | BIGTABLESIZE |
| Percentage Increase | BIGTABLEPCTINCREASE |
| Tablespace | BIGTABLESPACE |

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  major_vers      number(5)          NOT NULL,
  minor_vers      number(5)          NOT NULL,
  pd_version      number(7)          ,
  pa_comment      varchar2(128)      NULL,
  pa_event        number(5)          ,
  pa_date         date                ,
  pa_user         varchar2(24)       )
```

| Column | Description |
|-------------------------|--|
| <code>node_id</code> | ID of the node that this audit event is stored on, as defined in the nodes table. |
| <code>proc_id</code> | Procedure number of the procedure associated with this audit event, as defined in the proc_index table. |
| <code>major_vers</code> | Major version number of the version associated with this audit event, as defined in the proc_version table. |
| <code>minor_vers</code> | Minor version number of the version associated with this audit event, as defined in the proc_version table. |
| <code>pd_version</code> | Instance number of the procedure definition associated with this audit event, as defined in the proc_instance table. |
| <code>pa_comment</code> | Comment describing the audit event. |
| <code>pa_event</code> | The audit event that occurred. Either: Created (0), Updated (1), Released (2) or Withdrawn (3). |
| <code>pa_date</code> | 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 user_names 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 ¹ | node_id proc_id 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 | Tablespace |
|-------------------|----------------------------------|---------------|
| idx_proc_audit_fk | node_id proc_id pd_version | BIGINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---------------------|
| Initial | BIGTABLESIZE |
| Percentage Increase | BIGTABLEPCTINCREASE |
| Tablespace | BIGTABLESPACE |

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. |

| A row is... | When... |
|-------------|------------------------------------|
| 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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  pd_version       number(5)          NOT NULL,
  pd_type          number(1)          NOT NULL,
  pd_index         number(10)         NOT NULL,
  pd_size          number(10)         NOT NULL,
  pd_directory     varchar2(25)       NOT NULL,
  pd_file          varchar2(25)       NOT NULL,
  pd_data          long raw           NOT NULL)
```

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

| Column | Description |
|--------------|---|
| pd_index | <p>Index number into the set of rows that make up this procedure definition.</p> <p>If the procedure definition is longer than 30,000 bytes, multiple rows (in 30,000 byte chunks) are used to store the data. Each segment of the procedure definition data is uniquely identified by its pd_index value.</p> <p>Note:</p> <ul style="list-style-type: none"> If you have upgraded your system from one of the following versions, existing procedure definitions are divided into 2000 byte chunks: <ul style="list-style-type: none"> (UNIX Oracle) any Version i9.n-o iProcess Engine, or from a pre-Version 9.0-o(0.15) Process Engine. (Windows Oracle) any Version 9.0-o Process Engine. However, if a procedure definition is modified, the new instance is added using 30,000 byte chunks. |
| pd_size | <p>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.</p> |
| pd_directory | <p>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.</p> <p>If pd_type is 0 to 4, this field contains a hyphen.</p> |
| pd_file | <p>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.</p> <p>If pd_type is 0 to 4, this field contains a hyphen.</p> |
| pd_data | <p>Raw data for this (portion of the) procedure definition.</p> |

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

| Key Name | Column(s) | Index Tablespace |
|--------------|--|------------------|
| pk_proc_defn | proc_id pd_index pd_type node_id pd_version pd_directory pd_file | LARGEINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|---------------------------|----------------------------------|------------------------|
| fk_proc_defn ¹ | node_id proc_id 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 | Tablespace |
|------------------|---|-----------------|
| idx_proc_defn_fk | proc_id node_id pd_version | LARGEINDEXSPACE |
| idx_proc_defn | proc_id pd_version pd_type node_id | LARGEINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | LARGETABLESIZE |
| Percentage Increase | LARGETABLEPCTINCREASE |
| Tablespace | LARGETABLESPACE |

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  major_vers       number(5)          NOT NULL,
  minor_vers       number(5)          NOT NULL,
  pd_version       number(7)          NOT NULL,
  dead_name        varchar(32)        NOT NULL,
  event_name       varchar(32)        NOT NULL,
  dead_value       varchar(512)       NOT NULL)
```

| Column | Description |
|-------------------------|---|
| <code>node_id</code> | ID of the node that this procedure is defined on, as defined in the nodes table. |
| <code>proc_id</code> | Unique ID of this procedure, generated from the sequences table. |
| <code>major_vers</code> | The major version number of the procedure that this case belongs to, as defined in the proc_version table. |
| <code>minor_vers</code> | The minor version number of the procedure that this case belongs to, as defined in the proc_version table. |
| <code>pd_version</code> | Instance number of the procedure definition, as defined in the proc_instance 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> | The value of the case deadline. The value is specified in one of the following formats: <ul style="list-style-type: none"> If the case deadline is specified as a period, then the value is in the format: <pre>minutes^hours^days^weeks^months^years</pre> If the case deadline is specified as an expression, then the value is in the format: <pre>date expression^time expression</pre> |

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

| Key Name | Column(s) | Index Tablespace |
|------------------|---|------------------|
| pk_proc_deadline | node_id proc_id pd_version dead_name | BIGINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|-------------------------------|----------------------------------|------------------------|
| fk_proc_deadline ¹ | node_id proc_id 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 | Tablespace |
|----------------|----------------------------------|---------------|
| idx_proc_dl_fk | node_id proc_id pd_version | BIGINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---------------------|
| Initial | BIGTABLESIZE |
| Percentage Increase | BIGTABLEPCTINCREASE |
| Tablespace | BIGINDEXSPACE |

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. |

| A row is... | When... |
|-------------|----------------------------|
| 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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  major_vers       number(5)          NOT NULL,
  minor_vers       number(5)          NOT NULL,
  pd_version       number(7)          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 nodes table. |
| <code>proc_id</code> | ID of the procedure that this event belongs to, as defined in the proc_index table. |
| <code>major_vers</code> | Major version number of the procedure version that this case belongs to, as defined in the proc_version table. |
| <code>minor_vers</code> | Minor version number of the procedure version that this case belongs to, as defined in the proc_version table. |
| <code>pd_version</code> | Instance number of the procedure definition, as defined in the proc_instance 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"> • BeforePurge • BeforeClose • AfterClose • BeforeResurrect • AfterResurrect • BeforeSuspend • AfterSuspend • BeforeResume • AfterResume |

| 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) | Index Tablespace |
|---------------|---|------------------|
| pk_proc_event | node_id proc_id pd_version eventname | BIGINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|----------------------------|----------------------------------|------------------------|
| fk_proc_event ¹ | node_id proc_id pd_version | proc_instance |

1. This key enforces the DELETE CASCADE referential action.

Index The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-------------------|----------------------------------|---------------|
| idx_proc_event_fk | node_id proc_id pd_version | BIGINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---------------------|
| Initial | BIGTABLESIZE |
| Percentage Increase | BIGTABLEPCTINCREASE |
| Tablespace | BIGINDEXSPACE |

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

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

wqd_delta_subscriptions

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

Structure The `wqd_delta_subscriptions` table has the following structure:

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

| Column | Description |
|-----------------------------------|--|
| <code>wis_process_instance</code> | The instance of the WIS process that is responding to Work Queue Delta publication requests. |
| <code>queue_name</code> | The name of the work queue being monitored. |
| <code>wqdid</code> | The unique ID of the subscription. |
| <code>jms_topic_name</code> | 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) | Index Tablespace |
|--|--------------------|-------------------------------|
| <code>pk_wqd_delta_subscription</code> | <code>wqdid</code> | <code>MEDIUMINDEXSPACE</code> |

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 80](#)
- [Table Relationships, page 81](#)
- [pm_objects, page 82](#)
- [pm_objects_lock, page 85](#)
- [pmobjects_security, page 88](#)
- [proc_mgt_hierarchy, page 90](#)

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.



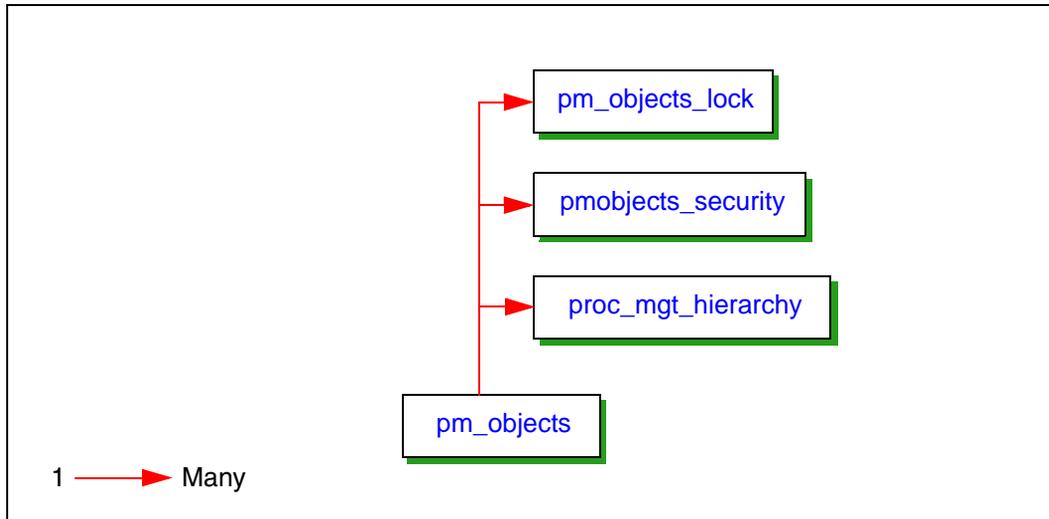
Shortcuts are *not* real procedure objects. They are simply placeholders that allow you to access a procedure object from different locations in the Procedure Management library. Data on shortcuts is only stored in the [proc_mgt_hierarchy](#) table.

Information about procedure versions is stored in other tables - see [Procedures on page 43](#) 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          varchar2(36)          NOT NULL,
  object_type          number(5)             NOT NULL,
  object_name          varchar2(64)         NOT NULL,
  version_major        number(5)            ,
  version_minor        number(5)            ,
  icon_mod_time        date                  NOT NULL,
  icon_binary          long raw              ,
  icon_size            number(8)           NOT NULL,
  object_url           varchar2(1000)       ,
  author               varchar2(64)        ,
  object_create_time   date                 NOT NULL,
  object_mod_time      date                 NOT NULL,
  release_id           varchar2(64)        ,
  security_all         number(1)           NOT NULL,
  proc_id              number(5)           NOT NULL,
  proc_status          number(5)            )
```

| Column | Description |
|----------------------------|--|
| <code>object_guid</code> | Globally unique, system-generated identifier for this procedure object. The row defining the Procedure Management library root has the value <code>ROOT_LIBRARY_GUID</code> . |
| <code>object_type</code> | Procedure object type. Either: library (0), procedure (1), sub-procedure (2) or sub-procedure parameter template (3). |
| <code>object_name</code> | Name of this procedure object. Note: The object's description (if defined) is also included as part of this field, in brackets following the name. |
| <code>version_major</code> | Major version number of the version associated with this procedure object, as defined in the proc_version table. This value is always 0 if the object is a library. |
| <code>version_minor</code> | Minor version number of the version associated with this procedure object, as defined in the proc_version table. This value is always 0 if the object is a library. |

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

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

| Key Name | Column(s) | Index Tablespace |
|---------------|-------------|------------------|
| pk_pm_objects | object_guid | SMALLINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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      varchar2(36)      NOT NULL,
  lck_state        number(5)          ,
  lck_owner        varchar2(24)      ,
  lck_time         date              )
```

| Column | Description |
|--------------------------|--|
| <code>object_guid</code> | ID for this procedure object, as defined in the pm_objects 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 user_names 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) | Index Tablespace |
|---------------------------------|--------------------------|---------------------------------|
| <code>pk_pm_objects_lock</code> | <code>object_guid</code> | SMALLINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|--|--------------------------|----------------------------|
| <code>fk_pm_objects_lock</code> ¹ | <code>object_guid</code> | pm_objects |

1. This key enforces the DELETE CASCADE referential action.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------|--------------------------------|
| Initial | SMALLTABLESIZE |

| Value | Definition |
|---------------------|---------------------------------------|
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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 Oracle as the background user.
2. In SQL*Plus, 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          varchar2(36)          NOT NULL,
  security_id          number(5)            NOT NULL,
  attrib_expr          varchar2(260)        NOT NULL,
  security_level       varchar2(8)         NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|----------------------|----------------------------|------------------|
| pk_pobjects_security | object_guid security_id | SMALLINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|-----------------------------------|-------------|----------------------------|
| fk_pobjects_security ¹ | object_guid | pm_objects |

1. This key enforces the DELETE CASCADE referential action.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------------------|
| Initial | <code>SMALLTABLESIZE</code> |
| Percentage Increase | <code>SMALLTABLEPCTINCREASE</code> |
| Tablespace | <code>SMALLTABLESPACE</code> |

Table Activity The `pobjects_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          varchar2(36)          NOT NULL,
  object_guid          varchar2(36)          NOT NULL,
  is_shortcut          number(1)             NOT NULL)
```

| Column | Description |
|--------------------------|--|
| <code>parent_guid</code> | ID for the parent library, as defined in the pm_objects table. |
| <code>object_guid</code> | ID for this procedure object, as defined in the pm_objects table. Note: If <code>is_shortcut</code> is 1, this value is the identifier of the procedure object pointed to by the shortcut. |
| <code>is_shortcut</code> | Flag that defines whether this hierarchy record is for a real procedure object (0) or for a shortcut (1). |

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

| Key Name | Column(s) | Index Tablespace |
|------------------------------------|--|---------------------------------|
| <code>pk_proc_mgt_hierarchy</code> | <code>parent_guid</code> <code>object_guid</code> | SMALLINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|--|--------------------------|----------------------------|
| <code>fk_proc_mgt_hierarchy¹</code> | <code>object_guid</code> | pm_objects |

1. This key enforces the DELETE CASCADE referential action.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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.

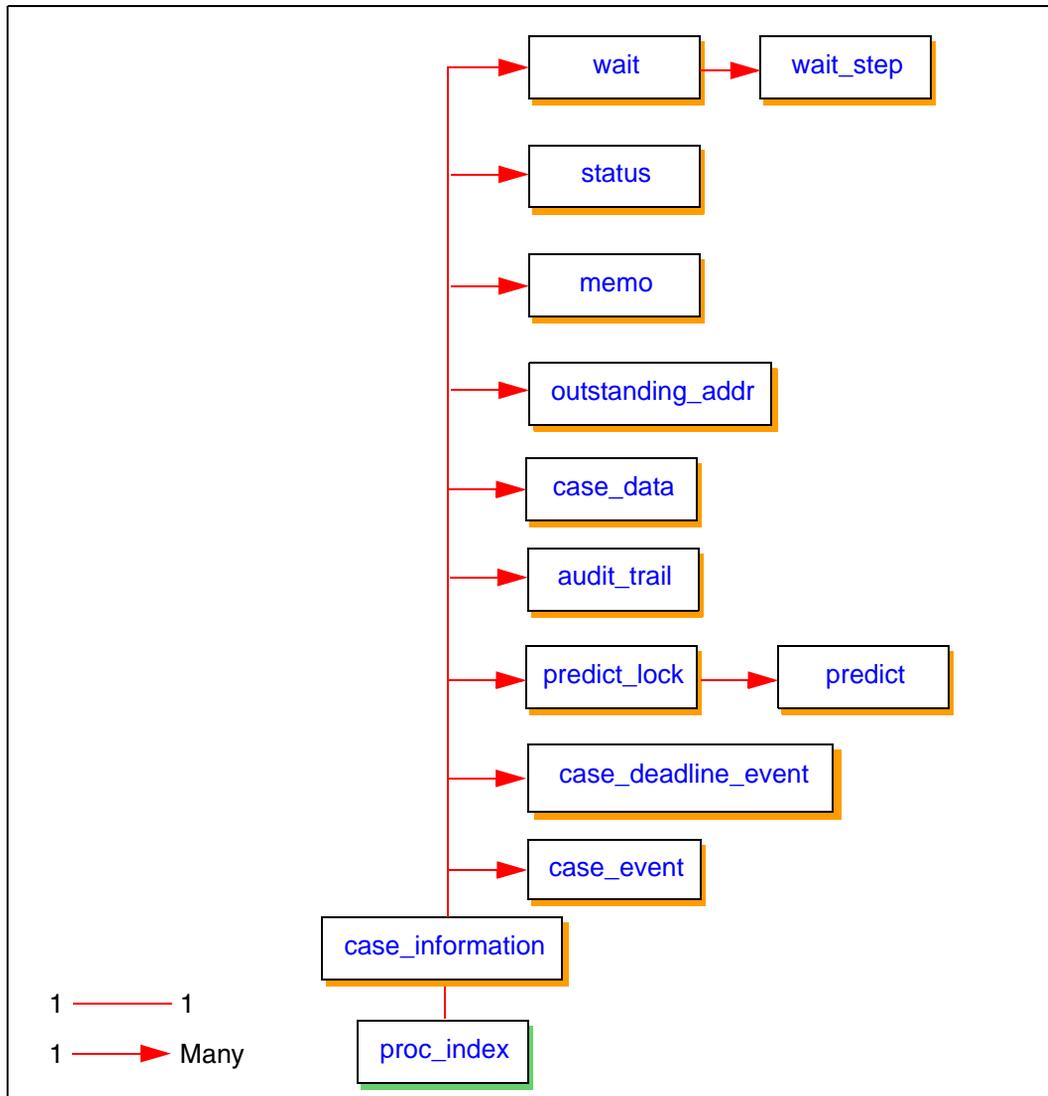
Topics

- [Table Relationships](#), page 94
- [case_information](#), page 95
- [outstanding_addr](#), page 99
- [wait](#), page 102
- [wait_step](#), page 104
- [status](#), page 106
- [case_data](#), page 108
- [audit_trail](#), page 111
- [memo](#), page 114
- [predict](#), page 117
- [predict_lock](#), page 121
- [case_deadline_event](#), page 123
- [case_event](#), page 126
- [casenum_gaps](#), page 130

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          number(5)           NOT NULL,
  proc_id          number(5)           NOT NULL,
  casenum         number(20)          NOT NULL,
  starter         varchar2(49)        NOT NULL,
  casedesc        varchar2(24)        NULL,
  procflags       number(5)           NOT NULL,
  next_deadline   date                ,
  is_subcase      number(1)           NOT NULL,
  is_dead         number(1)           NOT NULL,
  is_suspended    number(1)           NOT NULL,
  major_vers      number(5)           NOT NULL,
  minor_vers      number(5)           NOT NULL,
  proc_precedence number(5)           NOT NULL,
  started         date                NOT NULL,
  started_usecs   number(6)           NOT NULL)
```

| Column | Description |
|-----------------------|--|
| <code>node_id</code> | ID of the node that this case is hosted on, as defined in the nodes table. |
| <code>proc_id</code> | ID of the procedure that this case belongs to, as defined in the proc_index table. |
| <code>casenum</code> | Unique case number for this case, generated from the sequences table. Note: If the system has been upgraded from a Version 9 Process Engine, any cases that were started before the upgrade do not have a unique case number. (They have a number that is unique to that procedure.) |
| <code>starter</code> | Name of the user who started this case, as defined in the user_names table. |
| <code>casedesc</code> | Case description supplied when the case was started. |

| Column | Description |
|-----------------|--|
| procflags | <p>The procedure flags that were set at the time the case was started. For internal use only.</p> <p>Note: These flags are stored to allow consistent operation of the case if the procedure changes status during the lifetime of the case. For example, if the procedure is unreleased when the case is started, but changes to released before the case completes, the case can continue using the original procedure flags.</p> |
| next_deadline | <p>Date and time that the next deadline expires on this case. If no deadline is set this value appears as 12/31/3000 11:15:00 PM.</p> |
| is_subcase | <p>Flag that defines whether this case is a main case (0) or a sub-case (1).</p> |
| is_dead | <p>Flag that defines whether (1) or not (0) this case has completed.</p> |
| is_suspended | <p>Flag that defines whether (1) or not (0) the case is currently suspended (from a TIBCO iProcess Objects or SAL application).</p> |
| major_vers | <p>Major version number of the version of the procedure that this case belongs to, as defined in the proc_version table.</p> |
| minor_vers | <p>Minor version number of the version of the procedure that this case belongs to, as defined in the proc_version table.</p> |
| proc_precedence | <p>Stores the procedure precedence settings for opening sub-cases. One of:</p> <ul style="list-style-type: none"> • 32 - Released only. • 64 - Unreleased > Released. • 96 - Model > Released. • 128 - Unreleased > Model > Released. • 160 - Model > Unreleased > Released. |
| started | <p>Date and time that the case was started, to the resolution of a second.</p> <p>Note: The started_usec column can be combined with this column to provide resolution to a microsecond.</p> |
| started_usec | <p>Number of microseconds since the start of the <i>seconds</i> value specified in the started column.</p> |

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

| Key Name | Column(s) | Index Tablespace |
|---------------------|-------------------------------|------------------|
| pk_case_information | casenum proc_id node_id | LARGEINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|----------------------------------|--------------------|------------------------|
| fk_case_information ¹ | proc_id 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 | Tablespace |
|---------------------------------|---------------------------------------|-----------------|
| idx_case_information_fk | proc_id node_id | LARGEINDEXSPACE |
| idx_ci_casedesc_UC ¹ | UPPER(casedesc) casenum proc_id | LARGEINDEXSPACE |

1. Oracle function-based index, to facilitate case insensitive searches.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | LARGETABLESIZE |
| Percentage Increase | LARGETABLEPCTINCREASE |
| Tablespace | LARGETABLESPACE |

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

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

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

outstanding_addr

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

Structure The `outstanding_addr` table has the following structure:

```
TABLE outstanding_addr (
  node_id          number(5)           NOT NULL,
  proc_id          number(5)           NOT NULL,
  casenum         number(20)          NOT NULL,
  sentdate        date                 NOT NULL,
  deadline        number(1)           NOT NULL,
  deadline_expired number(1)           NOT NULL,
  sub_procedure   number(1)           NOT NULL,
  deaddate        date                 NOT NULL,
  stepname        varchar2(8)         NOT NULL,
  user_name       varchar2(64)        NOT NULL,
  reqid          number(20)          NOT NULL,
  item_suspended number(1)           NOT NULL,
  item_withdrawn number(1)           NOT NULL,
  array_idx      number(5)           NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|---------------------|------------------|------------------|
| pk_outstanding_addr | reqid node_id | LARGEINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|----------------------------------|-------------------------------|------------------------|
| fk_outstanding_addr ¹ | casenum proc_id node_id | case_information |

1. This key enforces the DELETE CASCADE referential action.

Indexes The following indexes are defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-------------------------|-------------------------------|-----------------|
| idx_outstanding_addr_fk | casenum proc_id node_id | LARGEINDEXSPACE |
| idx_deadline_date | deaddate | LARGEINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | LARGETABLESIZE |
| Percentage Increase | LARGETABLEPCTINCREASE |
| Tablespace | LARGETABLESPACE |

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

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

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

wait

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

Structure The `wait` table has the following structure:

```
TABLE wait (
  wait_id          number(10)          NOT NULL,
  node_id          number(5)           NOT NULL,
  proc_id          number(5)           NOT NULL,
  casenum         number(20)          NOT NULL,
  parentstep      varchar2(8)         NULL,
  expression       varchar2(200)      NULL,
  type            number(1)           NOT NULL)
```

| Column | Description |
|-------------------------|---|
| <code>wait_id</code> | Unique ID for this wait, generated from the sequences table. |
| <code>node_id</code> | ID of the node that this wait is hosted on, as defined in the nodes table. |
| <code>proc_id</code> | ID of the procedure that this wait belongs to, as defined in the proc_index table. |
| <code>casenum</code> | Case number that this wait belongs to, as defined in the case_information table. |
| <code>parentstep</code> | Step name of the parent step for this wait. |
| <code>expression</code> | <i>Not used. Reserved for possible future use.</i> |
| <code>type</code> | 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) | Index Tablespace |
|----------------------|--|-------------------------------|
| <code>pk_wait</code> | <code>wait_id</code> <code>node_id</code> | BIGINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|----------------------|-------------------------------|------------------------|
| fk_wait ¹ | node_id proc_id casenum | case_information |

1. This key enforces the DELETE CASCADE referential action.

Indexes The following indexes are defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-------------|-------------------------------|---------------|
| idx_wait_fk | casenum proc_id node_id | BIGINDEXSPACE |
| idx_wait | casenum proc_id | BIGINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---------------------|
| Initial | BIGTABLESIZE |
| Percentage Increase | BIGTABLEPCTINCREASE |
| Tablespace | BIGTABLESPACE |

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  wait_id          number(10)         NOT NULL,
  step_id          number(9)          NOT NULL)
```

| Column | Description |
|----------------------|--|
| <code>node_id</code> | ID of the node that the wait is hosted on, as defined in the nodes table. |
| <code>proc_id</code> | ID of the procedure that the wait belongs to, as defined in the proc_index table. |
| <code>wait_id</code> | ID of the wait, as defined in the wait_step 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 status table. |

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

| Key Name | Column(s) | Index Tablespace |
|---------------------------|--|-------------------------------|
| <code>pk_wait_step</code> | <code>wait_id</code> <code>step_id</code> | BIGINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|--|--|------------------------|
| <code>fk_wait_step</code> ¹ | <code>node_id</code> <code>wait_id</code> | wait |

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

Indexes The following indexes are defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|------------------|--------------------|---------------|
| idx_wait_step_fk | wait_id node_id | BIGINDEXSPACE |
| idx_wait_step | wait_id proc_id | BIGINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---------------------|
| Initial | BIGTABLESIZE |
| Percentage Increase | BIGTABLEPCTINCREASE |
| Tablespace | BIGTABLESPACE |

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  casenum         number(20)         NOT NULL,
  step_num        number(4)          NOT NULL,
  step_status     number(1)          NOT NULL)
```

| Column | Description |
|--------------------------|--|
| <code>node_id</code> | ID of the node that this step is hosted on, as defined in the nodes table. |
| <code>proc_id</code> | ID of the procedure that this step belongs to, as defined in the proc_index table. |
| <code>casenum</code> | Case number that this step belongs to, as defined in the case_information 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) | Index Tablespace |
|------------------------|---|--------------------------------|
| <code>pk_status</code> | <code>casenum</code> <code>proc_id</code> <code>step_num</code> <code>node_id</code> | HUGETABLESPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|-------------------------------------|--|----------------------------------|
| <code>fk_status</code> ¹ | <code>node_id</code> <code>proc_id</code> <code>casenum</code> | case_information |

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

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|---------------|-------------------------------|----------------|
| idx_status_fk | casenum proc_id node_id | HUGEINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | HUGETABLESIZE |
| Percentage Increase | HUGETABLEPCTINCREASE |
| Tablespace | HUGETABLESPACE |

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  casenum          number(20)         NOT NULL,
  field_name       varchar2(31)       NOT NULL,
  field_value      varchar2(255)      NULL,
  field_value_N    varchar2(255)      NULL)
```

| Column | Description |
|----------------------------|---|
| <code>node_id</code> | ID of the node that this field is hosted on, as defined in the nodes table. |
| <code>proc_id</code> | ID of the procedure that this field belongs to, as defined in the proc_index table. |
| <code>casenum</code> | Case number that this field belongs to, as defined in the case_information 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"> • Date values are stored as YYYY-MM-DD. • Numeric values are stored as padded strings. • Time and String values are not changed. <p>Note: This value is stored to make case data searching easier, so that the database can do simple string comparisons, instead of having to do type conversions.</p> <p>Case data can be normalized either when installing/upgrading the iProcess Engine, or by using the Case Data Normalization Utility - see <i>TIBCO iProcess Engine Administrator's Guide</i>.</p> |

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

| Key Name | Column(s) | Index Tablespace |
|--------------|---|-------------------|
| pk_case_data | casenum proc_id node_id field_name | MASSIVEINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|---------------------------|-------------------------------|------------------------|
| fk_case_data ¹ | node_id proc_id casenum | case_information |

1. This key enforces the DELETE CASCADE referential action.

Indexes The following indexes are defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|---|---|-------------------|
| idx_case_data_fk | casenum proc_id node_id | MASSIVEINDEXSPACE |
| idx_case_data_cnum_ procid_fnfv ¹ | field_name field_value_N casenum proc_id | MASSIVEINDEXSPACE |

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.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-------------------------|
| Initial | MASSIVETABLESIZE |
| Percentage Increase | MASSIVETABLEPCTINCREASE |
| Tablespace | MASSIVETABLESPACE |

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  casenum          number(20)         NOT NULL,
  type_id          number(3)          NOT NULL,
  audit_date       date               NOT NULL,
  stepdesc        varchar2(24)       NULL,
  user_name        varchar2(64)      NULL,
  stepname         varchar2(8)        NULL,
  audit_usecs      number(6)          NOT NULL,
  major_vers       number(5)          NOT NULL,
  minor_vers       number(5)          NOT NULL)
```

| Column | Description |
|-------------------------|--|
| <code>node_id</code> | ID of the node that this audit event is hosted on, as defined in the nodes table. |
| <code>proc_id</code> | ID of the procedure that this audit event belongs to, as defined in the proc_index table. |
| <code>casenum</code> | Case number that this audit event belongs to, as defined in the case_information table. |
| <code>type_id</code> | ID of the audit event that occurred. Either: <ul style="list-style-type: none"> a system-defined audit event (≤ 255), as defined in the <code>SWDIR\etc\language.lng\audit.mes</code> file. a custom, application-defined event (256-999), as defined in the <code>SWDIR\etc\language.lng\auditusr.mes</code> file. Note: 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. |
| <code>audit_date</code> | Date and time that this audit event occurred. <p>Note: The <code>audit_usecs</code> column can be combined with this column to provide resolution to a microsecond.</p> |

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

Primary Key None.

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

| Key Name | Column(s) | Referenced in Table... |
|-----------------------------|-------------------------------|------------------------|
| fk_audit_trail ¹ | node_id proc_id casenum | case_information |

1. This key enforces the DELETE CASCADE referential action.

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|--------------------|-------------------------------|-------------------|
| idx_audit_trail_fk | casenum proc_id node_id | MASSIVEINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-------------------------|
| Initial | MASSIVETABLESIZE |
| Percentage Increase | MASSIVETABLEPCTINCREASE |
| Tablespace | MASSIVETABLESPACE |

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          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  casenum          number(20)         NOT NULL,
  memo_id          number(4)          NOT NULL,
  memo_index       number(10)         NOT NULL,
  memo_size        number(10)         NOT NULL,
  memo_data        long raw           NOT NULL,
  array_idx        number(6)          NOT NULL)
```

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

Note:

- If you have upgraded your system from iProcess Engine one of the following versions, existing memos are divided into 2,000 byte chunks:
 - (UNIX Oracle) any Version i9.n-o iProcess Engine, or from a pre-Version 9.0-o(0.15) Process Engine.
 - (Windows Oracle) any Version 9.0-o Process Engine.
- However, if a memo is modified, the existing rows are deleted and then re-added using 30,000 byte chunks.

| Column | Description |
|-----------|--|
| memo_size | Size (in bytes) of the memo data for this row. |
| memo_data | Memo data. |
| array_idx | Either: <ul style="list-style-type: none"> The array element index number of the memo. -1, if the memo is not an array memo field. |

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

| Key Name | Column(s) | Index Tablespace |
|----------|---|------------------|
| pk_memo | casenum memo_id memo_index proc_id node_id array_idx | HUGEINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|----------------------|-------------------------------|------------------------|
| fk_memo ¹ | node_id proc_id casenum | case_information |

1. This key enforces the DELETE CASCADE referential action.

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-------------|-------------------------------|----------------|
| idx_memo_fk | casenum proc_id node_id | HUGEINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | HUGETABLESIZE |
| Percentage Increase | HUGETABLEPCTINCREASE |

| Value | Definition |
|------------|----------------|
| Tablespace | HUGETABLESPACE |

Table Activity

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

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

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          number(5)          NOT NULL,
  proc_num         number(5)          NOT NULL,
  case_num         number(20)         NOT NULL,
  parent_proc_num  number(5)          NOT NULL,
  parent_case_num  number(20)         NOT NULL,
  main_proc_num    number(5)          NOT NULL,
  main_case_num    number(20)         NOT NULL,
  step_name        varchar2(8)        NOT NULL,
  step_desc        varchar2(24)        NULL,
  step_desc2       varchar2(24)        NULL,
  step_addr        varchar2(49)        NOT NULL,
  step_durn_secs   number(10)          NULL,
  step_durn_usec   number(10)          NULL,
  step_start       date                NOT NULL,
  step_start_usec  number(10)          NOT NULL,
  step_end         date                NOT NULL,
  step_end_usec    number(10)          NOT NULL,
  field_name       varchar2(31)        NULL,
  field_value      varchar2(255)       NULL)
```

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

| Column | Description |
|------------------|--|
| parent_case_num | <p>Either:</p> <ul style="list-style-type: none"> ID of the parent case associated with this predicted work item, as defined in the case_information table, if case_num is a sub-case. 0, if this is a predicted work item in a future sub-case, rather than a currently outstanding sub-case, that was itself started from a predicted future sub-case. |
| main_proc_num | ID of the procedure associated with the main case that generated this predicted work item, as defined in the proc_index table. |
| main_case_num | ID of the main case that generated this predicted work item, as defined in the case_information table. |
| step_name | Stepname of the step associated with this predicted work item. |
| step_desc | Step description of the step associated with this predicted work item. |
| 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 | <p>Expected duration (in seconds) between this predicted work item being delivered to and released from the step_addr queue.</p> <p>Note: The step_durn_usecs column can be combined with this column to provide resolution to a microsecond.</p> |
| step_durn_usecs | Number of microseconds to be added to the value specified in the step_durn_secs column. |
| step_start | <p>Date and time that this predicted work item is expected to arrive in the step_addr queue, to the resolution of a second.</p> <p>Note: The step_start_usecs column can be combined with this column to provide resolution to a microsecond.</p> |
| step_start_usecs | Number of microseconds since the start of the seconds 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. Note: The step_end_usecs column can be combined with this column to provide resolution to a microsecond. |
| step_end_usecs | Number of microseconds since the start of the 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 ¹ | main_case_num main_proc_num node_id | predict_lock | case_num proc_num node_id |

1. This key enforces the DELETE CASCADE referential action.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |



By default, the predict and predict_lock tables use the SMALL macro values. If you intend to enable case prediction on your system, TIBCO recommends that you change these tables to use a larger value in line with the level of prediction activity you expect.

Table Activity

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

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

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

| A row is... | When... |
|-------------|---|
| added | background prediction is enabled on the iProcess Engine, and anything occurs that causes prediction data for a case to be calculated or recalculated. For example, when a case is started, a work item is kept or released, a deadline expires or an event occurs. Note: One row is added for each step in the procedure that can occur on the currently predicted path(s). |
| updated | never. |
| deleted | background prediction is enabled on the iProcess Engine, and anything occurs that causes prediction data for a case to be recalculated. For example, when a work item is kept or released, a deadline expires or an event occurs. Note: All rows for a given main case number are deleted for each step in the procedure that can no longer occur on the currently predicted path(s). |



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          number(5)          NOT NULL,
  proc_num         number(5)          NOT NULL,
  case_num         number(20)         NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|------------------------------|--|---------------------------------|
| <code>pk_predict_lock</code> | <code>node_id</code> <code>proc_num</code> <code>case_num</code> | SMALLINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... | Column(s) |
|---|--|----------------------------------|--|
| <code>fk_predict_loc</code> <code>k¹</code> | <code>node_id</code> <code>proc_num</code> <code>case_num</code> | case_information | <code>node_id</code> <code>proc_id</code> <code>casenum</code> |

1. This key enforces the DELETE CASCADE referential action.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |



By default, the `predict` and `predict_lock` tables use the `SMALL` macro values. If you intend to enable case prediction on your system, TIBCO recommends that you change these tables to use a larger value in line with the level of prediction activity you expect.

Table Activity

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

| A row is... | When... |
|-------------|---|
| added | background prediction is enabled on the iProcess Engine, and a case that has prediction enabled is started. Note: Case prediction can be enabled and disabled using the <code>ENABLE_CASE_PREDICTION</code> process attribute. See <i>TIBCO iProcess Engine Administrator's Guide</i> for more information. |
| 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          number(5)          NOT NULL,
  proc_id          number(5)          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 |
|-------------------------|--|
| <code>node_id</code> | ID of the node that this procedure is defined on, as defined in the nodes table. |
| <code>proc_id</code> | Unique ID of this procedure, generated from the sequences table. |
| <code>casenum</code> | The number of the case that this case deadline belongs to, as defined in the case_information table. |
| <code>dead_id</code> | For internal use only. This column is referenced from the <code>stepname</code> column in the outstanding_addr table. |
| <code>dead_name</code> | The name of the case deadline. |
| <code>event_name</code> | The name of the event step. |
| <code>dead_value</code> | The value of the case deadline when the case is running. The value is specified in one of the following formats: <ul style="list-style-type: none"> If the case deadline is specified as a period, then the value is in the format: <i>minutes^hours^days^weeks^months^years</i> If the case deadline is specified as an expression, then the value is in the format: <i>date expression^time expression</i> |

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

| Key Name | Column(s) | Index Tablespace |
|------------------|--|------------------|
| pk_case_dl_event | node_id proc_id casenum dead_id | BIGINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|-------------------------------|-------------------------------|------------------------|
| fk_case_dl_event ¹ | casenum proc_id node_id | case_information |

1. This key enforces the DELETE CASCADE referential action.

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|----------------|-------------------------------|---------------|
| idx_case_dl_fk | casenum proc_id node_id | BIGINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | LARGETABLESIZE |
| Percentage Increase | LARGETABLEPCTINCREASE |
| Tablespace | LARGETABLESPACE |

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

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

case_event

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

Structure The `case_event` table has the following structure:

```
TABLE case_event (
  node_id          number(5)          NOT NULL,
  proc_id          number(5)          NOT NULL,
  major_vers       number(5)          NOT NULL,
  minor_vers       number(5)          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 |
|-------------------------|--|
| <code>node_id</code> | ID of the node that this case is hosted on, as defined in the nodes table. |
| <code>proc_id</code> | ID of the procedure that this procedure event belongs to, as defined in the proc_index table. |
| <code>major_vers</code> | Major version number of the procedure version that this case belongs to, as defined in the proc_version table. |
| <code>minor_vers</code> | Minor version number of the procedure version that this case belongs to, as defined in the proc_version 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"> • BeforePurge • BeforeClose • AfterClose • BeforeResurrect • AfterResurrect • BeforeSuspend • AfterSuspend • BeforeResume • AfterResume |
| user_event_name | The name of the event step which you set for the procedure event. |
| casenum | ID of the case that this event belongs to, as defined in the case_information table. |
| state | <p>Flag that defines the state of the procedure event after the event is triggered. The meaning for each flag is:</p> <ul style="list-style-type: none"> • 2 the triggered event is in the processing state. • 3 the triggered event is finished. • 4 the triggered event is cancelled. • -1 the triggered event failed. |
| actionparameter | 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. |

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

| Key Name | Column(s) | Index Tablespace |
|---------------|--|------------------|
| pk_case_event | node_id proc_id casenum eventname | LARGETABLESPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|----------------------------|-------------------------------|------------------------|
| fk_case_event ¹ | casenum proc_id node_id | case_information |

1. This key enforces the DELETE CASCADE referential action.

Index The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-------------------|-------------------------------|-----------------|
| idx_case_event_fk | casenum proc_id node_id | LARGETABLESPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | LARGETABLESIZE |
| Percentage Increase | LARGETABLEPCTINCREASE |
| Tablespace | LARGETABLESPACE |

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. |

| A row is... | When... |
|-------------|---|
| 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 data-bbox="635 338 899 373">• The case is purged.<li data-bbox="635 385 1049 420">• The procedure event is finished.<li data-bbox="635 432 992 466">• The procedure event failed.<li data-bbox="635 479 1063 513">• The procedure event is cancelled. |

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      number(20)      NOT NULL,
  casenum_max      number(20)      NOT NULL)
```

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

Primary Key

The following primary key is defined for this table.

| Key Name | Column(s) | Index Tablespace |
|------------------------------|--------------------------|--------------------------------|
| <code>pk_casenum_gaps</code> | <code>casenum_min</code> | <code>MASSIVETABLESPACE</code> |

Foreign Keys

None.

Triggers

None.

Indexes

None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---|
| Initial | MASSIVETABLESIZE |
| Percentage Increase | MASSIVETABLEPCTINCREASE |
| Tablespace | MASSIVETABLESPACE |

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

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

See Also [CASENUM_FIND_GAPS](#)

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”).

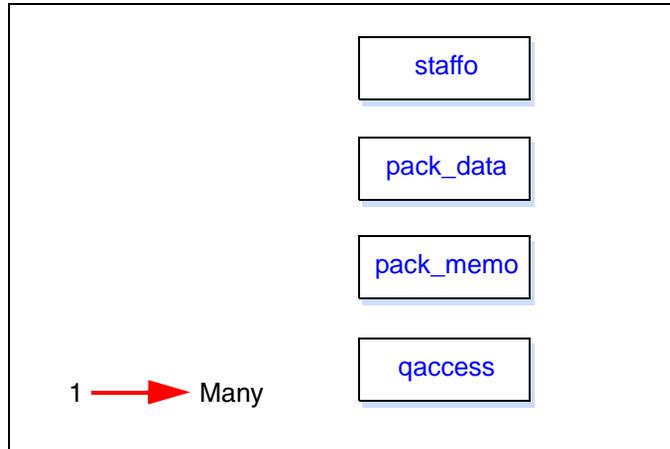
Topics

- [Table Relationships, page 134](#)
- [staffo, page 135](#)
- [pack_data, page 139](#)
- [pack_memo, page 141](#)
- [qaccess, page 144](#)

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(
  o_flags                number(5)           NULL,
  o_queueename           varchar2(24)        NULL,
  o_locker               varchar2(24)        NULL,
  o_username             varchar2(49)        NULL,
  o_startname            varchar2(49)        NULL,
  o_dirname              varchar2(12)        NOT NULL,
  o_dirdesc              varchar2(24)        NULL,
  o_procname             varchar2(8)NOT NULL,
  o_procdesc             varchar2(24)NULL,
  o_casedesc             varchar2(24)        NULL,
  o_casenum              number(20)         NULL,
  o_placeno              number(5)          NULL,
  o_dirflags             number(5)          NULL,
  o_procflags            number(5)          NULL,
  o_host                 varchar2(24)        NOT NULL,
  o_pnum                 number(5)          NOT NULL,
  o_pnumcount            number(5)          NOT NULL,
  o_caseptr              number(15)         NULL,
  o_reqidhost            number(24)         NOT NULL,
  o_reqid                number(20)NOT NULL,
  o_deadline             dateNULL,
  o_reqstamp             date               NULL,
  o_qparam1              varchar2(24)        NULL,
  o_qparam2              varchar2(24)        NULL,
  o_qparam3              varchar2(12)        NULL,
  o_qparam4              varchar2(12)        NULL,
  o_itempriority         varchar2(24)        NULL,
  o_priority_changed     dateNULL,
  o_majorvers            number(5)NOT NULL,
  o_minorvers            number(5)NOT NULL)
```

| Column | Description |
|---------------------------|--|
| <code>o_flags</code> | Flags associated with this work item. For internal use only. |
| <code>o_queueename</code> | Queue name of the user or group queue that contains this work item, as defined in the user_names table. |
| <code>o_locker</code> | Name of the user who has locked the queue (if it is locked), as defined in the user_names table. Note: This column is not written to or updated unless the <code>WIS_WRITELOCKS</code> parameter in the <code>SWDIR\etc\staffcfg</code> file is set. |

| Column | Description |
|-------------|--|
| o_username | Queue name of the user or group queue that contains this work item, as defined in the user_names table. |
| o_startname | Username of the user who started the case that this work item belongs to, as defined in the user_names table. |
| o_dirname | Step name of the step that generated this work item. |
| o_dirdesc | Step description of the step that generated this work item. |
| o_procname | Procedure name of the procedure that generated this work item, as defined in the proc_index table. |
| o_procdesc | Procedure description of the procedure that generated this work item, as defined in the proc_index table. |
| o_casedesc | Case description of the case that this work item belongs to, as defined in the case_information table. |
| o_casenum | Case number of the case that this work item belongs to, as defined in the case_information table. |
| o_placeno | Step mark number. For internal use only. |
| o_dirflags | Step flags. For internal use only. |
| o_procfags | Procedure flags. For internal use only. |
| o_host | ID of the node that this work item is associated with, as defined in the nodes table. |
| o_pnum | Procedure number of the procedure that generated this work item, as defined in the proc_index table. |
| o_pnumcount | Version count of procedure. For internal use only. |
| o_caseptr | Case control record number. For internal use only. |
| o_reqidhost | Nodename of the node where the o_reqid is generated, as defined in the nodes table. |
| o_reqid | Unique ID for this work item, generated from the sequences table. |
| o_deadline | Date and time that the deadline (if defined) expires on this work item. If no deadline is set this value appears as 12/31/3000 11:15:00 PM. |

| Column | Description |
|--------------------|--|
| o_reqstamp | Timestamp when this work item was delivered to the queue. |
| o_qparam1 | Value of work queue parameter 1 for this work item. |
| o_qparam2 | Value of work queue parameter 2 for this work item. |
| o_qparam3 | Value of work queue parameter 3 for this work item. |
| o_qparam4 | Value of work queue parameter 4 for this work item. |
| o_itempriority | <p>Priority definition for this work item, in the format: <i>base:increment:number:period:type</i></p> <p>where:</p> <ul style="list-style-type: none"> • <i>base</i> is the base priority value for this work item. • <i>increment</i> is the amount that will be added to the item's priority value whenever the <i>period</i> expires. • <i>number</i> is the number of increments that will be added to the item's priority value. • <i>period</i> is the time period, in the units specified in <i>type</i>, which must expire before the item's priority value is incremented. • <i>type</i> is the unit of measure of the <i>period</i>, either "M" or "m" for minutes, "H" or "h" for hours or "D" or "d" for days. |
| o_priority_changed | Timestamp when the priority value for this work item was last changed. |
| o_majorvers | Major version number of the procedure that generated this work item, as defined in the proc_version table. |
| o_minorvers | Minor version number of the procedure that generated this work item, as defined in the proc_version table. |

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

| Key Name | Column(s) | Index Tablespace |
|-----------|------------------------|------------------|
| pk_staffo | o_reqid o_reqidhost | HUGEINDEXSPACE |

Foreign Keys None.

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|------------|-------------------|----------------|
| idx_staffo | o_queueName | HUGEINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | HUGETABLESIZE |
| Percentage Increase | HUGETABLEPCTINCREASE |
| Tablespace | HUGETABLESPACE |

Table Activity The `staffo` table contains one row for every outstanding step on the system. Rows are added, updated and deleted in the following situations.

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

pack_data

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

When a work item is sent out to a queue, field data is copied from the `case_data` table to the `pack_data` table. The client uses the field values in the `pack_data` table to fill out the form correctly. When the form is kept any changed fields are updated in the `pack_data` 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          number(20)          NOT NULL,
  node_id        number(5)           NOT NULL,
  proc_id        number(5)           NOT NULL,
  casenum        number(20)          NOT NULL,
  field_name     varchar2(31)        NOT NULL,
  field_value    varchar2(255)       NULL,
  field_flags    number(5)           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. Note: 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) | Index Tablespace |
|--------------|--------------------------------|------------------|
| pk_pack_data | reqid node_id field_name | HUGEINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | HUGETABLESIZE |
| Percentage Increase | HUGETABLEPCTINCREASE |
| Tablespace | HUGETABLESPACE |

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

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

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

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          number(20)NOT NULL,
  node_id       number(5)NOT NULL,
  proc_id       number(5)NOT NULL,
  casenum       number(20)NOT NULL,
  memo_id       number(4)NOT NULL,
  memo_index    number(10)NOT NULL,
  memo_size     number(10)NOT NULL,
  memo_data     long rawNOT NULL,
  array_idx     number(6)NOT NULL)
```

| Column | Description |
|----------------------|---|
| <code>reqid</code> | ID of the work item that this memo belongs to, as defined in the <code>staffo</code> table. |
| <code>node_id</code> | ID of the node that this memo is associated with, as defined in the <code>nodes</code> table. |
| <code>proc_id</code> | Number of the procedure that this memo belongs to, as defined in the <code>proc_index</code> table. |
| <code>casenum</code> | Case number that this memo belongs to, as defined in the <code>case_information</code> table. |
| <code>memo_id</code> | Unique (for this case) ID of this memo. |

| Column | Description |
|------------|--|
| memo_index | <p>Index number into the set of rows that make up this memo. If a memo is longer than 30,000 bytes multiple rows (in 30,000 byte chunks) are used to store the memo data. Each segment of the memo data is uniquely identified by its memo_index value.</p> <p>Note:</p> <ul style="list-style-type: none"> If you have upgraded your system from one of the following versions, existing memos are divided into 2000 byte chunks: <ul style="list-style-type: none"> (UNIX Oracle) any Version i9.n-o iProcess Engine, or from a pre-Version 9.0-o(0.15) Process Engine. (Windows Oracle) any Version 9.0-o Process Engine. However, if a memo is modified, the existing rows are deleted and then re-added using 30,000 byte chunks. |
| memo_size | Size (in bytes) of the memo data for this row. |
| memo_data | Memo data. |
| array_idx | <p>Either:</p> <ul style="list-style-type: none"> The array element index number of the memo. -1, if the memo is not an array memo field. |

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

| Key Name | Column(s) | Index Tablespace |
|--------------|---|------------------|
| pk_pack_memo | reqid node_id casenum memo_id memo_index array_idx | HUGEINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | HUGETABLESIZE |
| Percentage Increase | HUGETABLEPCTINCREASE |
| Tablespace | HUGETABLESPACE |

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

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

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

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          varchar2(64)          NOT NULL,
  access_type        varchar2(8)           NOT NULL,
  queue_name         varchar2(51)         NOT NULL,
  access_str         varchar2(1024)       NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|-------------------------|---|-------------------------------|
| <code>pk_qaccess</code> | <code>user_name</code> <code>queue_name</code> <code>access_type</code> | <code>MEDIUMINDEXSPACE</code> |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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. |

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

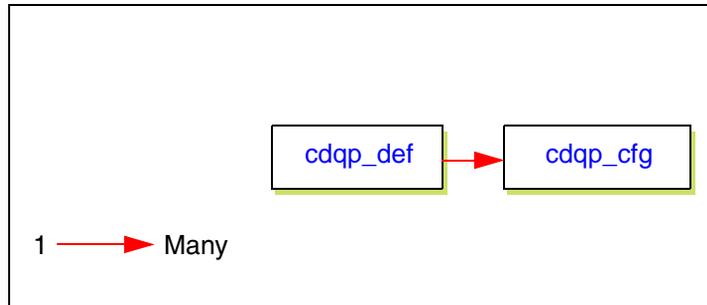
Topics

- [Table Relationships, page 148](#)
- [cdqp_def, page 149](#)
- [cdqp_cfg, page 151](#)

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          number(10)          NOT NULL,
  field_name     varchar2(31)        NOT NULL,
  data_size      number(5)           NOT NULL,
  description    varchar2(40)        NOT NULL,
  is_predict     number(1)           NOT NULL)
```

| Column | Description |
|--------------------------|--|
| <code>def_id</code> | Unique identifier for this CDQP, generated from the sequences table. |
| <code>field_name</code> | Name of the iProcess field assigned to this CDQP, as defined in the case_data 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) | Index Tablespace |
|--------------------------|---------------------|----------------------------------|
| <code>pk_cdqp_def</code> | <code>def_id</code> | MEDIUMINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|--|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |

| Value | Definition |
|------------|------------------|
| Tablespace | MEDIUMTABLESPACE |

Table Activity

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

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

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

cdqp_cfg

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

Structure The `cdqp_cfg` table has the following structure:

```
TABLE cdqp_cfg (
  cfg_id          number(10)          NOT NULL,
  def_id          number(10)          NOT NULL,
  queue_name     varchar2(48)        NOT NULL)
```

| Column | Description |
|-------------------------|---|
| <code>cfg_id</code> | Unique ID for this CDQP/queue mapping generated from the sequences table. |
| <code>def_id</code> | ID of the CDQP that is mapped to the <code>queue_name</code> queue, as defined in the cdqp_def 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 user_names table. |

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

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

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

| Key Name | Column(s) | Referenced in Table... |
|---------------------------------------|---------------------|--------------------------|
| <code>fk_cdqp_cfg</code> ¹ | <code>def_id</code> | cdqp_def |

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

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|------------------------------|---------------------|----------------------------------|
| <code>idx_cdqp_cfg_fk</code> | <code>def_id</code> | MEDIUMINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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.

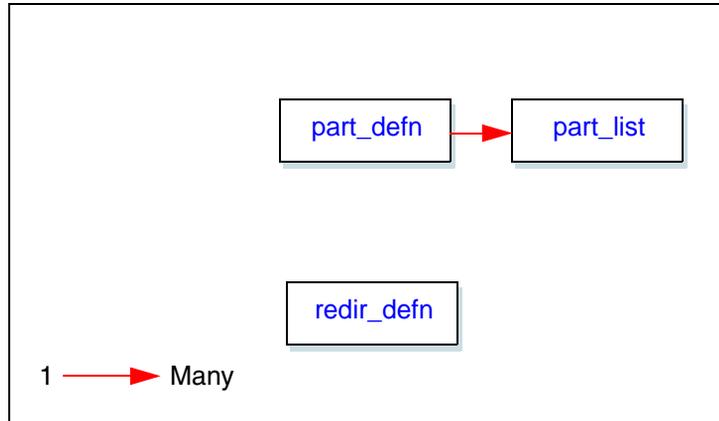
Topics

- [Table Relationships, page 154](#)
- [part_defn, page 155](#)
- [part_list, page 157](#)
- [redir_defn, page 159](#)

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          number(5)          NOT NULL,
  queue_name       varchar2(24)       NOT NULL,
  days_mask        varchar2(7)        NOT NULL,
  start_time       number(4)          NOT NULL,
  end_time         number(4)          NOT NULL,
  style            varchar2(24)       NULL,
  start_date       number(7)          NOT NULL,
  end_date         number(7)          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 <code>user_names</code> table. |
| <code>days_mask</code> | Days of the week that users can participate in the specified <code>queue_name</code> . For example, <code>-TWT-SS</code> 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) | Index Tablespace |
|---------------------------|----------------------|-------------------------------|
| <code>pk_part_defn</code> | <code>part_id</code> | <code>MEDIUMINDEXSPACE</code> |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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          number(5)          NOT NULL,
  user_name        varchar2(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 part_defn 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 user_names 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_list¹</code> | <code>part_id</code> | part_defn |

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

Indexes The following index is defined for this table.

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

Storage The following `STORAGE` values are defined for this table.

| Value | Definition |
|---------------------|--|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

Table Activity

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

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

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

redir_defn

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

Structure The `redir_defn` table has the following structure:

```
TABLE redir_defn (
  redir_id          number(5)          NOT NULL,
  start_time        number(4)          NOT NULL,
  start_date        number(7)          NOT NULL,
  end_time          number(4)          NOT NULL,
  end_date          number(7)          NOT NULL,
  queue_name        varchar2(24)       NOT NULL,
  destination       varchar2(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 user_names table. |
| <code>destination</code> | Name of the queue to which work items are to be redirected, as defined in the user_names table. |

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

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

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-------------------------------------|
| Initial | <code>MEDIUMTABLESIZE</code> |
| Percentage Increase | <code>MEDIUMTABLEPCTINCREASE</code> |
| Tablespace | <code>MEDIUMTABLESPACE</code> |

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 162](#)
- [flag_table, page 163](#)
- [version, page 166](#)

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          number(5)          NOT NULL,
  area_locked      number(1)          NOT NULL,
  area_changed     number(1)          NOT NULL,
  move_req         number(1)          NOT NULL,
  user_name        varchar2(64)       NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|---------------|-----------|------------------|
| pk_flag_table | area_id | TINYINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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

The table is populated when the iProcess Engine is installed.

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

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

version

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

Structure The `version` table has the following structure:

```
TABLE version (
  version_type      varchar2(20)      NOT NULL,
  version_value     number(6)        NOT NULL)
```

| Column | Description |
|----------------------------|--|
| <code>version_type</code> | Data type: either <code>cdqp</code> or <code>user</code> . |
| <code>version_value</code> | Number that is incremented whenever the data is changed. |

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

| Key Name | Column(s) | Index Tablespace |
|-------------------------|---------------------------|-----------------------------|
| <code>pk_version</code> | <code>version_type</code> | <code>TINYINDEXSPACE</code> |

Foreign Keys None.

Indexes None.

Storage The following `STORAGE` values are defined for this table.

| Value | Definition |
|---------------------|-----------------------------------|
| Initial | <code>TINYTABLESIZE</code> |
| Percentage Increase | <code>TINYTABLEPCTINCREASE</code> |
| Tablespace | <code>TINYTABLESPACE</code> |

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 flag_table 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 170](#)
- [Table Relationships, page 171](#)
- [user_names, page 172](#)
- [user_attrib, page 174](#)
- [user_settings, page 177](#)
- [user_values, page 178](#)
- [user_memb, page 181](#)
- [leavers, page 183](#)
- [tsys_user_names, page 185](#)
- [tsys_user_attrib, page 186](#)
- [tsys_user_values, page 187](#)
- [tsys_user_memb, page 188](#)

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).

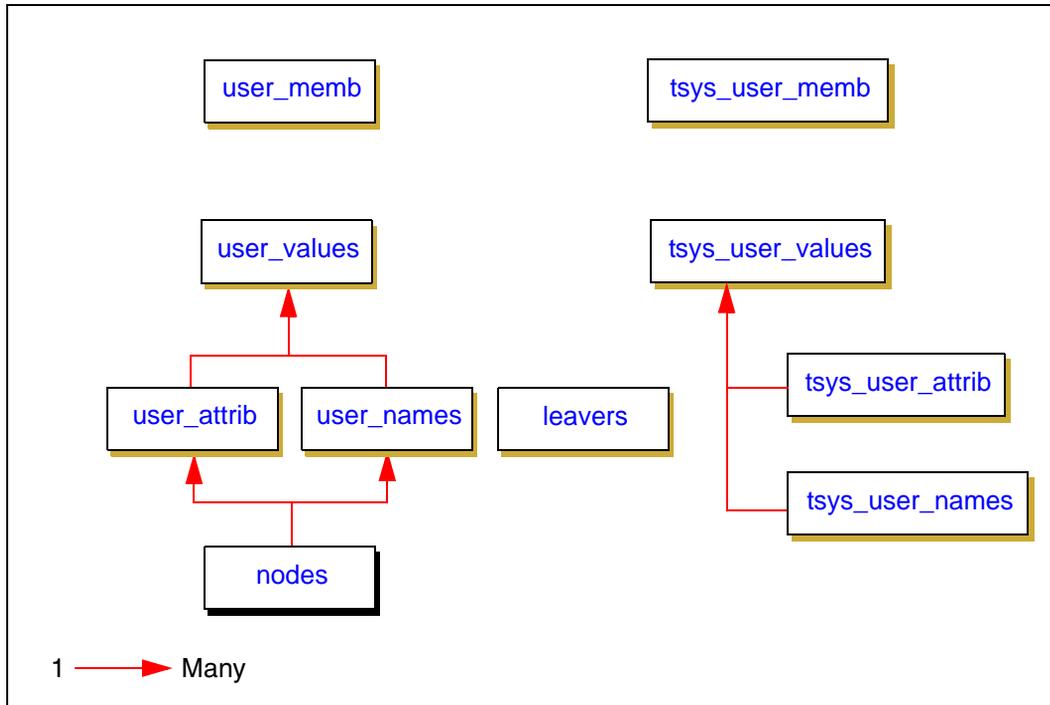
The `user_` tables are purged and rewritten with the updated information from the `tsys_user_` tables when a Move System Information is performed - *if* the [flag_table](#) indicates that the appropriate data area has been modified.

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          number(5)          NOT NULL,
  user_id          number(6)          NOT NULL,
  user_name        varchar2(64)       NOT NULL,
  user_type        varchar2(1)        NOT NULL)
```

| Column | Description |
|------------------------|--|
| <code>node_id</code> | ID of the node that this (user or group) queue is registered on, as defined in the nodes table. |
| <code>user_id</code> | Unique ID for this (user or group) queue. Note: Users and groups have separate ID sequences, as defined in the <code>user_type</code> column, so both a user and a group can have the same <code>user_id</code> value. |
| <code>user_name</code> | Name of this (user or group) queue. |
| <code>user_type</code> | Queue type: user (U) or group (G). |

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

| Key Name | Column(s) | Index Tablespace |
|----------------------------|--|----------------------------------|
| <code>pk_user_names</code> | <code>user_id</code> <code>user_type</code> <code>node_id</code> | MEDIUMINDEXSPACE |

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

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

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

Indexes The following indexes are defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-------------------|-------------------|------------------|
| idx_user_names_fk | node_id | MEDIUMINDEXSPACE |
| idx_user_names | user_name | MEDIUMINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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 | a Move System Information is performed, if the flag_table indicates that the Users data area has been modified. Note: The table is purged and rewritten using the values from the tsys_user_names table. |
| updated | never. |
| deleted | a Move System Information is performed, if the flag_table indicates that the Users data area has been modified. Note: The table is purged and rewritten using the values from the tsys_user_names table. |

user_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          number(5)          NOT NULL,
  attribute_id     number(5)          NOT NULL,
  attribute_name   varchar2(15)     NOT NULL,
  attribute_type   varchar2(1)      NOT NULL)
```

| Column | Description |
|-----------------------------|--|
| <code>node_id</code> | ID of the node that this attribute is defined on, as defined in the nodes table. |
| <code>attribute_id</code> | Unique ID for this attribute. |
| <code>attribute_name</code> | Name of this attribute. |
| <code>attribute_type</code> | 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) | Index Tablespace |
|-----------------------------|---|---------------------------------|
| <code>pk_user_attrib</code> | <code>attribute_id</code> <code>node_id</code> | SMALLINDEXSPACE |

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

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

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

Indexes The following indexes are defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|---------------------------------|-----------------------------|---------------------------------|
| <code>idx_user_attrib_fk</code> | <code>node_id</code> | SMALLINDEXSPACE |
| <code>idx_user_attrib</code> | <code>attribute_name</code> | SMALLINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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 | a Move System Information is performed, if the flag_table indicates that the Users data area has been modified. Note: The table is purged and rewritten using the values from the tsys_user_names table. |
| updated | never. |
| deleted | a Move System Information is performed, if the flag_table indicates that the Users data area has been modified. Note: The table is purged and rewritten using the values from the tsys_user_names table. |

user_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          number(5)          NOT NULL,
  user_name        varchar2(64)       NULL,
  setting_key      varchar2(64)       NULL,
  setting_value    varchar2(2048)     NULL
```

| Column | Description |
|----------------------------|---|
| <code>node_id</code> | ID of the node that this attribute is defined on, as defined in the nodes table. |
| <code>user_name</code> | The name of the user whose preferences these are, as defined in the user_names table. |
| <code>setting_key</code> | The key to identify a particular setting. |
| <code>setting_value</code> | 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) | Index Tablespace |
|-------------------------------|--|-------------------------------|
| <code>pk_user_settings</code> | <code>node_id</code> <code>user_name</code> <code>setting_key</code> <code>setting_value</code> | BIGINDEXSPACE |

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          number(5)          NOT NULL,
  user_id          number(6)          NOT NULL,
  attribute_id     number(5)          NOT NULL,
  attribute_value  varchar2(24)      NULL,
  user_type       varchar2(1)        NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|----------------|---|------------------|
| pk_user_values | user_id attribute_id user_type node_id | BIGINDEXSPACE |

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

| Key Name ¹ | Column(s) | Referenced in Table... |
|-----------------------|---------------------------------|------------------------|
| fk_user_values1 | node_id user_id user_type | user_names |
| fk_user_values2 | attribute_id 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 | Tablespace |
|---------------------|---------------------------------|---------------|
| idx_user_values_fk1 | node_id user_id user_type | BIGINDEXSPACE |
| idx_user_values_fk2 | attribute_id node_id | BIGINDEXSPACE |
| idx_user_values | attribute_id | BIGINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---------------------|
| Initial | BIGTABLESIZE |
| Percentage Increase | BIGTABLEPCTINCREASE |
| Tablespace | BIGTABLESPACE |

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 flag_table indicates that the Users data area has been modified. Note: The table is purged and rewritten using the values from the tsys_user_names table. |
| updated | never. |
| deleted | a Move System Information is performed, if the flag_table indicates that the Users data area has been modified. Note: The table is purged and rewritten using the values from the tsys_user_names table. |

user_memb

The `user_memb` table defines users' membership of groups.

Structure The `user_memb` table has the following structure:

```
TABLE user_memb (
  node_id          number(5)          NOT NULL,
  user_id          number(6)          NOT NULL,
  group_id         number(6)          NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|---------------------------|---|-------------------------------|
| <code>pk_user_memb</code> | <code>user_id</code> <code>node_id</code> <code>group_id</code> | BIGINDEXSPACE |

Foreign Keys None.

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|----------------------------|---|-------------------------------|
| <code>idx_user_memb</code> | <code>user_id</code> <code>group_id</code> | BIGINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-------------------------------------|
| Initial | BIGTABLESIZE |
| Percentage Increase | BIGTABLEPCTINCREASE |

| Value | Definition |
|------------|---------------|
| Tablespace | BIGTABLESPACE |

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

leavers

The leavers table stores information about the recently deleted users.

Structure The leavers table has the following structure:

```
TABLE leavers (
  node_id          number(5)          NOT NULL,
  user_name        varchar2(64)       NOT NULL,
  destination      varchar2(64)       NOT NULL,
  timestamp        number(20)         NOT NULL,
  status           number(2)          NOT NULL)
```

| Column | Description |
|-------------|---|
| node_id | ID of the node that this (user or group) queue is registered on, as defined in the nodes table. |
| user_name | Name of this deleted user. |
| destination | Description of this deleted user. |
| timestamp | When the current status is set. |
| status | Status of the redirection performed on the leaver. One of the following values: <ul style="list-style-type: none"> 0 (LEAVER_WILL_BE_REDIRECTED) The leaver will be redirected. 1 (LEAVER_IS_BEING_REDIRECTED) The leaver is being redirected. 2 (LEAVER_FINISH_REDIRECTION) The leaver has been redirected. |

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

| Key Name | Column(s) | Index Tablespace |
|------------|----------------------|------------------|
| pk_leavers | user_name node_id | MEDIUMINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

Table Activity The leavers table contains one row for each recently deleted user. Rows are added, updated, and deleted in the following situations.

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

tsys_user_names

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

- The primary key name is `pk_tsys_user_names`.
- No 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_values`.
- The foreign key names are `fk_tsys_user_values1` and `fk_tsys_user_values2`.
- The index names are `idx_tsys_user_values_fk1` and `idx_tsys_user_values_fk2`.
- The table is purged and rewritten when a user edits user data, either by saving changes made in the User Manager utility in the TIBCO iProcess Administrator, importing data with `SWDIR\bin\swutil USERINFO`, or by using TIBCO iProcess Objects. (The `flag_table` is also updated to indicate that the Users data area has been modified.)

tsys_user_memb

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

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

Chapter 13 **Roles**

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

Topics

- [About Roles, page 190](#)
- [Table Relationships, page 191](#)
- [role_users, page 192](#)
- [tsys_role_users, page 194](#)

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).

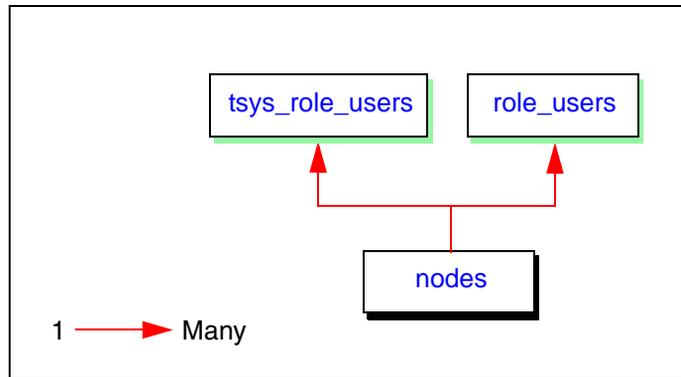
When a Move System Information is performed, if the [tsys_role_users](#) table has been changed, the [role_users](#) table is purged and rewritten with the updated information from the [tsys_role_users](#) table.

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          number(5)          NOT NULL,
  role_id          number(5)          NOT NULL,
  role_name        varchar2(15)      NOT NULL,
  usernode_name    varchar2(49)      NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|----------------------------|--|----------------------------------|
| <code>pk_role_users</code> | <code>role_id</code> <code>node_id</code> | MEDIUMINDEXSPACE |

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

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

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

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|--------------------------------|----------------------|----------------------------------|
| <code>idx_role_users_fk</code> | <code>node_id</code> | MEDIUMINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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

tsys_role_users

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

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

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 an Oracle table.

Topics

- [About TIBCO iProcess Tables, page 196](#)
- [Table Relationships, page 197](#)
- [dbs_names, page 198](#)
- [dbs_fields, page 200](#)
- [dbs_values, page 202](#)
- [tsys_dbs_names, page 204](#)
- [tsys_dbs_fields, page 205](#)
- [tsys_dbs_values, page 206](#)
- [str_dbs_names, page 207](#)
- [str_dbs_fields, page 208](#)
- [ttmp_dbs_names, page 209](#)
- [ttmp_dbs_fields, page 210](#)
- [ttmp_dbs_values, page 211](#)

About TIBCO iProcess Tables

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

| Prefix | Description |
|-----------|--|
| db*_ | Hold the main system data on <i>installed</i> TIBCO iProcess tables, which iProcess references during normal operation. |
| str_db*_ | Hold the main system data on <i>uninstalled</i> TIBCO iProcess tables, which iProcess references during normal operation. Note: There is no <code>str_db*_values</code> 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).

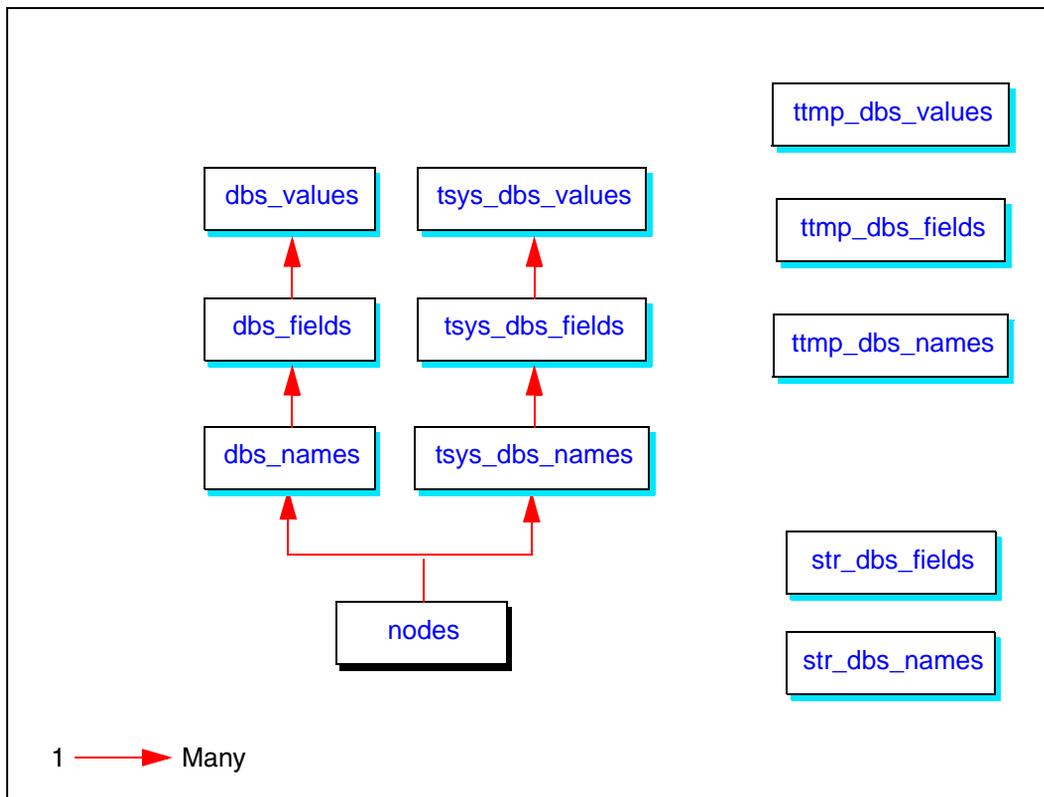
When a Move System Information is performed, if the `tsys_db*_` tables have been changed, the `db*_` and/or `str_db*_` tables are purged and rewritten with the updated information from the `tsys_db*_` tables.

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          number(5)          NOT NULL,
  dbs_id           number(5)          NOT NULL,
  dbs_name         varchar2(15)      NOT NULL)
```

| Column | Description |
|-----------------------|--|
| <code>node_id</code> | ID of the node that this iProcess Engine table is defined on, as defined in the nodes table. |
| <code>dbs_id</code> | Unique ID for this iProcess Engine table. |
| <code>dbs_name</code> | Name of this iProcess Engine table. |

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

| Key Name | Column(s) | Index Tablespace |
|---------------------------|---|------------------------------|
| <code>pk_dbs_names</code> | <code>dbs_id</code> <code>node_id</code> | <code>SMALLINDEXSPACE</code> |

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

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

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

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-------------------------------|----------------------|------------------------------|
| <code>idx_dbs_names_fk</code> | <code>node_id</code> | <code>SMALLINDEXSPACE</code> |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|---------------------------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

Table Activity The `dbms_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 flag_table indicates that the Tables data area has been modified. Note: The table is purged and rewritten using the values from the tsys_dbms_names table. |
| updated | never. |
| deleted | a Move System Information is performed, if the flag_table indicates that the Tables data area has been modified. Note: The table is purged and rewritten using the values from the tsys_dbms_names table. |

dbs_fields

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

Structure The `dbs_fields` table has the following structure:

```
TABLE dbs_fields (
  node_id          number(5)          NOT NULL,
  dbs_id           number(5)          NOT NULL,
  field_id         number(5)          NOT NULL,
  field_name       varchar2(15)      NOT NULL,
  field_type       varchar2(1)       NOT NULL,
  field_length     number(5)          NOT NULL,
  field_decimals   number(5)          NOT NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|----------------------------|--|----------------------------------|
| <code>pk_dbs_fields</code> | <code>dbs_id</code> <code>field_id</code> <code>node_id</code> | MEDIUMINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|--|---|---------------------------|
| <code>fk_dbs_fields¹</code> | <code>node_id</code> <code>dbs_id</code> | dbs_names |

1. This key enforces the DELETE CASCADE referential action.

Indexes The following indexes are defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-------------------|--------------------|------------------|
| idx_dbs_fields_fk | dbs_id node_id | MEDIUMINDEXSPACE |
| idx_dbs_fields | field_id dbs_id | MEDIUMINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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

dbs_values

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

Structure The `dbs_values` table has the following structure:

```
TABLE dbs_values (
  node_id          number(5)          NOT NULL,
  dbs_id           number(5)          NOT NULL,
  record_id        number(5)          NOT NULL,
  field_id         number(5)          NOT NULL,
  field_value      varchar2(30)      NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|----------------------------|--|-------------------------------|
| <code>pk_dbs_values</code> | <code>dbs_id</code> <code>record_id</code> <code>field_id</code> <code>node_id</code> | <code>MEDIUMINDEXSPACE</code> |

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

| Key Name | Column(s) | Referenced in Table... |
|---|--|----------------------------|
| <code>fk_dbs_values</code> ¹ | <code>dbs_id</code> <code>field_id</code> <code>node_id</code> | dbs_fields |

1. This key enforces the DELETE CASCADE referential action.

Indexes The following indexes are defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-------------------|---------------------------------|------------------|
| idx_dbs_values_fk | dbs_id field_id node_id | MEDIUMINDEXSPACE |
| idx_dbs_values | record_id field_id dbs_id | MEDIUMINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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 <code>flag_table</code> indicates that the Tables data area has been modified. Note: The table is purged and rewritten using the values from the <code>tsys_dbs_values</code> table. |
| updated | never. |
| deleted | a Move System Information is performed, if the <code>flag_table</code> indicates that the Tables data area has been modified. Note: The table is purged and rewritten using the values from the <code>tsys_dbs_values</code> 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_tsys_dbs_names_fk`.
- The table is purged and rewritten when a user edits iProcess Engine table data, either by saving changes made in the Table Manager utility in the TIBCO iProcess Administrator, modifying data with `SWDIR\bin\swutil IMPORT` or `DELTA`, 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_tsys_dbs_fields_fk` and `idx_tsys_dbs_fields`.
- The table is purged and rewritten when a user edits iProcess Engine table data, either by saving changes made in the Table Manager utility in the TIBCO iProcess Administrator, modifying data with `SWDIR\bin\swutil IMPORT` or `DELTA`, 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_tsys_dbs_values_fk` and `idx_tsys_dbs_values`.
- The table is purged and rewritten when a user edits iProcess Engine table data, either by saving changes made in the Table Manager utility in the TIBCO iProcess Administrator, modifying data with `SWDIR\bin\swutil IMPORT` or `DELTA`, 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 214](#)
- [Table Relationships, page 215](#)
- [list_names, page 216](#)
- [list_values, page 218](#)
- [tsys_list_names, page 220](#)
- [tsys_list_values, page 221](#)
- [ttmp_list_names, page 222](#)
- [ttmp_list_values, page 223](#)

About Lists

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

| Prefix | Description |
|-------------------------|---|
| <code>list_</code> | Hold the main system data on iProcess lists, which iProcess Engine references during normal operation. |
| <code>tsys_list_</code> | Hold a copy of the main system data, containing users' edits that have not yet been released for use by the system. |
| <code>ttmp_list_</code> | 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).

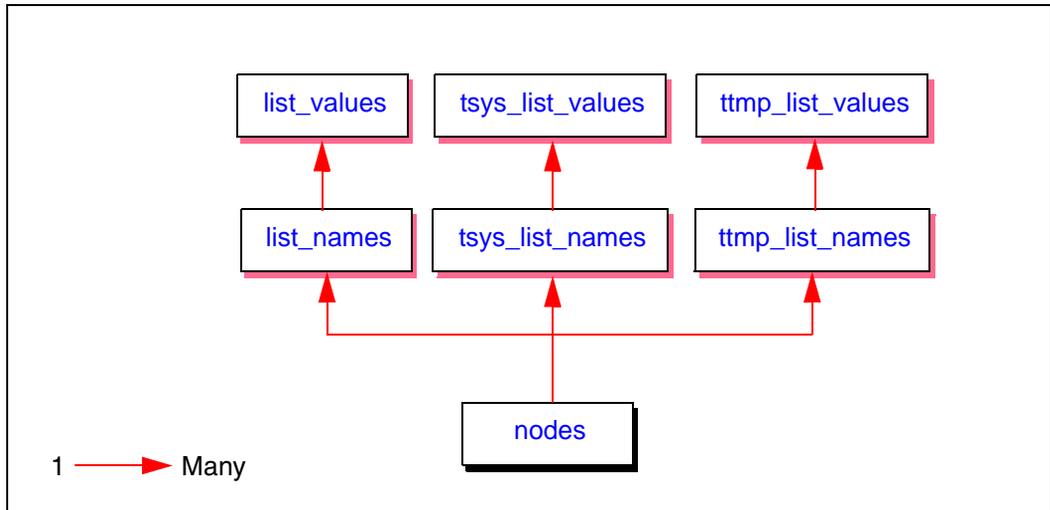
The `list_` tables are purged and rewritten with the updated information from the `tsys_list_` tables when a Move System Information is performed - *if* the [flag_table](#) indicates that the Lists data area has been modified

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          number(5)          NOT NULL,
  list_id          number(5)          NOT NULL,
  list_name        varchar2(15)      NOT NULL,
  list_type        varchar2(1)       NOT NULL,
  list_length      number(5)         NOT NULL,
  list_decimals    number(5)         NOT NULL)
```

| Column | Description |
|----------------------------|--|
| <code>node_id</code> | ID of the node that this list is stored on, as defined in the nodes table. |
| <code>list_id</code> | Unique ID for this list. |
| <code>list_name</code> | Name of this list. |
| <code>list_type</code> | List type: Either ASCII (A), Numeric (R), Date (D) or Time (T). |
| <code>list_length</code> | Length of this list, in characters. |
| <code>list_decimals</code> | 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) | Index Tablespace |
|----------------------------|--|------------------------------|
| <code>pk_list_names</code> | <code>list_id</code> <code>node_id</code> | <code>SMALLINDEXSPACE</code> |

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

| Key Name | Column(s) | Referenced in Table... |
|--|----------------------|------------------------|
| <code>fk_list_names¹</code> | <code>node_id</code> | nodes |

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

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|-------------------|-------------------|-----------------|
| idx_list_names_fk | node_id | SMALLINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|-----------------------|
| Initial | SMALLTABLESIZE |
| Percentage Increase | SMALLTABLEPCTINCREASE |
| Tablespace | SMALLTABLESPACE |

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 flag_table indicates that the Lists data area has been modified. Note: The table is purged and rewritten using the values from the tsys_list_names table. |
| updated | never. |
| deleted | a Move System Information is performed, if the flag_table indicates that the Lists data area has been modified. Note: The table is purged and rewritten using the values from the tsys_list_names 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          number(5)          NOT NULL,
  list_id          number(5)          NOT NULL,
  record_id       number(5)          NOT NULL,
  list_value      varchar2(30)       NULL)
```

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

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

| Key Name | Column(s) | Index Tablespace |
|-----------------------------|--|----------------------------------|
| <code>pk_list_values</code> | <code>list_id</code> <code>record_id</code> <code>node_id</code> | MEDIUMINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|--|--|----------------------------|
| <code>fk_list_values</code> ¹ | <code>node_id</code> <code>list_id</code> | list_names |

1. This key enforces the DELETE CASCADE referential action.

Indexes The following index is defined for this table.

| Index Name | Column(s) Indexed | Tablespace |
|--------------------|--------------------|------------------|
| idx_list_values_fk | list_id node_id | MEDIUMINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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 <code>flag_table</code> indicates that the Lists data area has been modified. Note: The table is purged and rewritten using the values from the <code>tsys_list_values</code> table. |
| updated | never. |
| deleted | a Move System Information is performed, if the <code>flag_table</code> indicates that the Lists data area has been modified. Note: The table is purged and rewritten using the values from the <code>tsys_list_values</code> 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_tsys_list_names_fk`.
- The table is purged and rewritten when a user edits iProcess list data, either by saving changes made in the List Manager utility in the TIBCO iProcess Administrator, modifying data with `SWDIR\bin\swutil IMPORT`, or by using TIBCO iProcess Objects. (The `flag_table` is also updated to indicate that the Lists data area has been modified.)

tsys_list_values

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

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

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_tmp_list_names_fk.
- In most situations the number of rows in the table should be zero.

ttmp_list_values

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

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

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.

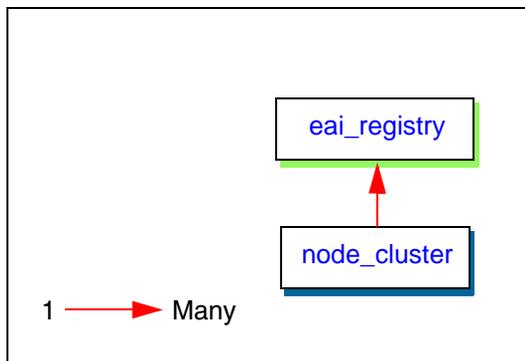
Topics

- [Table Relationships, page 226](#)
- [eai_registry, page 227](#)

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          varchar2(20)          NOT NULL,
  logical_machine_id number(5)           NOT NULL,
  release_version   varchar2(32)         NOT NULL,
  plugin_library    varchar2(256)        NOT NULL,
  init_params       varchar2(1024)       NULL)
```

| 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"> EAIDB EAI Database EATSCR EAI Script EAIWEBSERVICES EAI Web Services |
| <code>logical_machine_id</code> | ID of the computer that this iProcess server plug-in is installed on, as defined in the node_cluster table. Note: If a node cluster architecture is in use, the iProcess server plug-in must be installed on each server in the cluster that is configured to run a background process. |
| <code>release_version</code> | Version number of this iProcess server plug-in (for example, i10.0-o(3.0)). |
| <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) | Index Tablespace |
|------------------------------|--|------------------|
| <code>pk_eai_registry</code> | <code>eai_type</code> <code>logical_machine_id</code> | TINYINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|------------------------------|--------------------|------------------------|
| fk_eai_registry ¹ | 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 | Tablespace |
|---------------------|--------------------|----------------|
| idx_eai_registry_fk | logical_machine_id | TINYINDEXSPACE |

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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*.

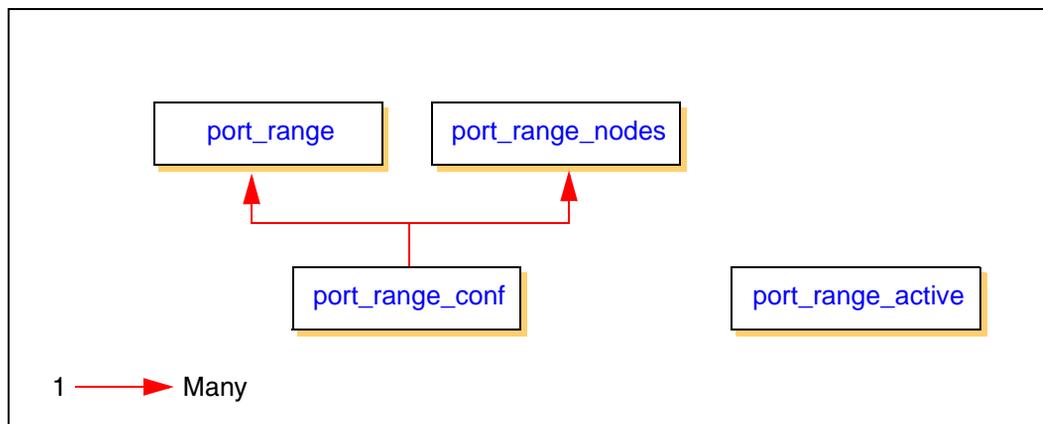
Topics

- [Table Relationships](#), page 230
- [port_range](#), page 231
- [port_range_active](#), page 234
- [port_range_conf](#), page 236
- [port_range_nodes](#), page 238

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      number(10)      NOT NULL,
  slot_number        number(10)      NOT NULL,
  rpc_number         number(10)      NOT NULL,
  port_number        number(10)      NOT NULL,
  status             number(10)      NOT NULL,
  logical_machine_id number(10)      NULL,
  logical_process_name varchar2(10)  NULL,
  logical_process_instance number(10) NULL)
```

| Column | Description |
|---------------------------------|---|
| <code>port_range_id</code> | Unique ID of the port range configuration that this port/RPC number belongs to, as defined in the port_range_conf table. |
| <code>slot_number</code> | Internal slot in memory used by this port/RPC number. |
| <code>rpc_number</code> | RPC number. |
| <code>port_number</code> | Port number. |
| <code>status</code> | 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"> • -2 Reserved for future use. • -1 Unobtainable. (A process tried to use the port but found that it was already in use.) • 0 Unallocated. • 1 Allocated to the process defined by the <code>logical_machine_id</code>, <code>logical_process_name</code> and <code>logical_process_instance</code> columns. |
| <code>logical_machine_id</code> | Either: <ul style="list-style-type: none"> • ID of the server where the process instance that this port/RPC number has been allocated to runs, as defined in the node_cluster table. • 0, if the port/RPC number has not been allocated to a process. |

| 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) | Index Tablespace |
|---------------|-------------------------------|------------------|
| pk_port_range | port_range_id, slot_number | TINYINDEXSPACE |

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

| Key Name | Column(s) | Referenced in Table... |
|---------------|---------------|------------------------|
| fk_port_range | port_range_id | port_range_conf |

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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

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

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

| A row is... | When... |
|-------------|---|
| deleted | a user deletes a port range configuration, that is, a record in the port_range_conf table, using the <i>SWDIR\util\swadm</i> utility. |

port_range_active

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



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 varchar2(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 node_cluster table. |
| <code>logical_process_name</code> | Logical name of this process instance. Note: See "Administering iProcess Engine Server Processes" in <i>TIBCO iProcess Engine Administrator's Guide</i> for a list of logical process names. |
| <code>logical_process_instance</code> | 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) | Index Tablespace |
|-----------------------------------|---|-----------------------------|
| <code>pk_port_range_active</code> | <code>logical_machine_id</code> , <code>logical_process_name</code> , <code>logical_process_instance</code> | <code>TINYINDEXSPACE</code> |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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 <code>port_range</code> 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"> 0 No Port or RPC ranging. A process uses the next available port number assigned by the operating system, and an RPC number based on the process ID. 1 Port ranging. A process uses a port number allocated from within the defined range, and an RPC number based on the process ID. 2 RPC ranging. A process uses the next available port number assigned by the operating system, and an RPC number allocated from within the defined range. 3 Port and RPC ranging. A process uses both a port number and an RPC number allocated from within the defined ranges. |
| <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) | Index Tablespace |
|--------------------|---------------|------------------|
| pk_port_range_conf | port_range_id | TINYINDEXSPACE |

Foreign Keys None.

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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 port_range_conf table. |
| <code>logical_machine_id</code> | ID of the server using this port range configuration, as defined in the node_cluster table. |

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

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

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> | port_range_conf |

Indexes None.

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|----------------------|
| Initial | TINYTABLESIZE |
| Percentage Increase | TINYTABLEPCTINCREASE |
| Tablespace | TINYTABLESPACE |

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

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

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

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    number(5)NOT NULL,
  logical_process_insta number(5)NOT NULL,
  queue_name            varchar(24)NOT NULL,
  total_items           number(20)NULL,
  last_cache_time       number(20)NULL,
  new_items             number(20)NULL,
  deadline_items        number(20)NULL,
  urgent_items          number(20)NULL,
  redir_queue_name      varchar(24)NULL,
  is_cached             number(1)NOT NULL,
  is_group              number(1)NOT NULL,
  is_test               number(1)NOT NULL,
  is_redirected         number(1)NOT NULL,
  is_disabled           number(1)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 node_cluster table. |
| <code>logical_process_insta nce</code> | ID of the instance of the WIS process that is handling this work queue, as defined in the process_config table. |
| <code>queue_name</code> | Name of the work queue. |
| <code>total_items</code> | Total number of items in this work queue. Note: When the iProcess Engine starts up the WIS processes use this value to determine whether or not each work queue should be cached. See "Configuring When WIS Processes Cache Their Queues" in <i>TIBCO iProcess Engine Administrator's Guide</i> for more information. |
| <code>last_cache_time</code> | Either: <ul style="list-style-type: none"> Time taken to cache the work queue (in milliseconds) when it was last cached, either when the WIS process first started up or when the work queue was first accessed. -1, if the work queue has not yet been cached. |

| 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"> the name of the work queue that this queue is currently being redirected to, if the queue is currently being redirected (<code>is_redirected = 1</code>). empty, if the queue is currently not being redirected (<code>is_redirected = 0</code>). |
| <code>is_cached</code> | Indicates whether the queue is currently cached by the WIS process. Either: <ul style="list-style-type: none"> 1, if the queue is cached. 0, if the queue is not cached. |
| <code>is_group</code> | Indicates whether the queue is a Group queue. Either: <ul style="list-style-type: none"> 1, if the queue is a Group queue. 0, if the queue is a User queue. |
| <code>is_test</code> | Indicates whether the queue is a Test queue. Either: <ul style="list-style-type: none"> 1, if the queue is a Test queue. 0, otherwise. |
| <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"> 1, if the queue is currently redirected. 0, otherwise. |
| <code>is_disabled</code> | Indicates whether the queue is disabled. Either: <ul style="list-style-type: none"> 1, if the queue is currently disabled. 0, otherwise. |

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

| Key Name | Column(s) | Index Tablespace |
|--------------|-----------------------|------------------|
| pk_wqs_index | queue_name is_test | MEDIUMINDEXSPACE |

Foreign Keys None

Indexes None

Storage The following STORAGE values are defined for this table.

| Value | Definition |
|---------------------|------------------------|
| Initial | MEDIUMTABLESIZE |
| Percentage Increase | MEDIUMTABLEPCTINCREASE |
| Tablespace | MEDIUMTABLESPACE |

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

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

| A row is... | When... |
|-------------|--|
| added | a new work queue is allocated to a WIS process by the WQS process. |
| updated | <ul style="list-style-type: none"> an existing work queue is re-allocated to a different WIS process by the WQS process. a MOVESYSINFO has been processed by the WQS process. the update thread in the WQS process writes the contents of the WQS/WIS shared memory to the database. This update occurs every WQS_PERSIST_SHMEM seconds. <p>Note: 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 (`init2Kora.sql` on UNIX, `init2Kora_tok.sql` on Windows).

Appendix B STORAGE Configuration Macro Values



STORAGE values are only defined if the database is *created* when installing a Version i10.0-o(2.0) or later iProcess Engine. They are not defined if the database is created for an earlier version and subsequently upgraded to Version i10.0-o(2.0) or later.

This appendix describes the real values that are assigned to the STORAGE configuration macros defined for each table in the iProcess database schema.

The real values are defined in three tablesizes files:

- *Default* values are defined in the tablesizes file. Using this file creates a small database (approximately 50Mb) that is suitable for benchmarking or development purposes
- *Medium* values are defined in the tablesizes `tablesizes.med` file. Using this file creates a medium database.
- *Large* values are defined in the tablesizes `tablesizes.large` file. Using this file creates a large database.

When you install the iProcess Engine, you choose which one of these tablesizes files to use. The installer (`swinstall`) replaces the configuration macros with the appropriate values in the `init2Kora.sql` script, which is used to create the iProcess database schema.

For more detailed information about this process, and how you can customize it to your particular requirements, see "Configure the Initial Size and Layout of the iProcess Schema" in *TIBCO iProcess Engine Installation Guide*.

The tables in the following sections define the real values that are assigned to each STORAGE configuration macro, according to:

- the *SIZE* value assigned to the specific table, as defined earlier in this guide.
- the tablesizes file used during installation.

TABLESIZE

The *SIZETABLESIZE* macros define the initial size (in extents) of iProcess schema tables.

| Macro | Default | Medium | Large |
|------------------|---------|---------|----------|
| TINYTABLESIZE | 1K | 1K | 1K |
| SMALLTABLESIZE | 2K | 10K | 20K |
| MEDIUMTABLESIZE | 40K | 90K | 90K |
| BIGTABLESIZE | 200K | 1200K | 4000K |
| LARGETABLESIZE | 500K | 15000K | 150000K |
| HUGETABLESIZE | 500K | 50000K | 1100000K |
| MASSIVETABLESIZE | 1M | 600000K | 1500000K |

TABLEPCTINCREASE

The *SIZEPCTINCREASE* macros define the percentage increase to be applied when growing iProcess schema tables.

| Macro | Default | Medium | Large |
|-------------------------|---------|--------|-------|
| TINYTABLEPCTINCREASE | 0 | 0 | 0 |
| SMALLTABLEPCTINCREASE | 20 | 20 | 20 |
| MEDIUMTABLEPCTINCREASE | 50 | 50 | 50 |
| BIGTABLEPCTINCREASE | 50 | 50 | 50 |
| LARGETABLEPCTINCREASE | 50 | 100 | 50 |
| HUGETABLEPCTINCREASE | 50 | 100 | 50 |
| MASSIVETABLEPCTINCREASE | 50 | 100 | 50 |

INDEXSIZE

This macro is not currently used. It is reserved for possible future use.

TABLESPACE

The *SIZETABLESPACE* macros define the tablespaces used by iProcess schema tables.

| Macro | Default | Medium | Large |
|-------------------|----------|----------|----------|
| TINYTABLESPACE | staffwar | staffwar | staffwar |
| SMALLTABLESPACE | staffwar | staffwar | staffwar |
| MEDIUMTABLESPACE | staffwar | staffwar | staffwar |
| BIGTABLESPACE | staffwar | staffwar | staffwar |
| LARGETABLESPACE | staffwar | staffwar | staffwar |
| HUGETABLESPACE | staffwar | staffwar | staffwar |
| MASSIVETABLESPACE | staffwar | staffwar | staffwar |

INDEXSPACE

The *SIZEINDEXSPACE* macros define the tablespaces used by iProcess schema primary keys and indexes.

| Macro | Default | Medium | Large |
|-------------------|----------|----------|----------|
| TINYINDEXSPACE | staffwar | staffwar | staffwar |
| SMALLINDEXSPACE | staffwar | staffwar | staffwar |
| MEDIUMINDEXSPACE | staffwar | staffwar | staffwar |
| BIGINDEXSPACE | staffwar | staffwar | staffwar |
| LARGEINDEXSPACE | staffwar | staffwar | staffwar |
| HUGEINDEXSPACE | staffwar | staffwar | staffwar |
| MASSIVEINDEXSPACE | staffwar | staffwar | staffwar |

AQTABLESPACE

The AQTABLESPACE macro defines the tablespace used by the AQ tables that provide the underlying message queuing system used by the iProcess message queues. See [Oracle AQ Queue Tables and Queues on page 32](#) for more information.

| Macro | Default | Medium | Large |
|--------------|----------|----------|----------|
| AQTABLESPACE | staffwar | staffwar | staffwar |

Appendix C **SSOLite Stored Procedures**

This appendix describes the SSOLite stored procedures package.

Topics

- [Overview, page 258](#)
- [Using SSOLite Stored Procedures, page 259](#)
- [Data Procedures, page 264](#)
- [Command Procedures, page 271](#)
- [Control Procedures, page 294](#)
- [Debug Procedures, page 304](#)

Overview

The SSOLite package 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 259](#)
- [Transactional Processing, page 259](#)
- [Handling Exceptions, page 259](#)
- [Processing Queues, page 261](#)
- [Prioritizing Messages, page 262](#)

Processing Asynchronous Message

SSOLite stored procedures work by sending a message to a database queue, which is processed by one or more background (BG) processes. This means that:

- there is a short delay between an SSOLite stored procedure completing and the BG process processing the instruction.
- even if an SSOLite procedure has completed successfully, the instruction that is processed by the BG may still fail.

Transactional Processing

The BG process will not process any instructions issued by SSOLite stored procedures until the SSOLite transaction has been committed. You can therefore scope transactions according to the requirements of your particular application:

- A transaction can be defined as a single instruction, such as a case start. (If the call to `SW_CASESTART` succeeds then a commit is immediately performed.)
- Several instructions can be processed as part of a single transaction. For example, a transaction can add pack data, issue an event, add more pack data and then start several cases, and is only committed if all these operations complete successfully.

Handling Exceptions

SSOLite stored procedures raise an Oracle error in the range -20000 to -20020 if any procedure fails. Note that:

- The error text is always preceded by the string (SWERROR).

- Each error has a unique ID, which is displayed at the end of the error text.

It is the application's responsibility to handle any such database exceptions, and issue a rollback if appropriate.

The following table describes the different errors (and their unique IDs) that may be returned by the SSOLite stored procedures.



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 | Oracle Error | Error Text |
|------------------------|--------------|--|
| SW_GET_NODE_DETAILS | -20001 | Node details not found in database (ID:001007) |
| SW_SET_QUEUE_NAME | 20002 | MBox Queue Name(s) not found in database (ID:001008) |
| SW_GET_PROCEDURE | -20003 | Procedure details not found in database for procedure name= <i>proc_name</i> (ID:001010) |
| | -20000 | Procedure version not found in database for procedure name= <i>proc_name</i> , Case Num= <i>case_num</i> (ID:001011) |
| | -20004 | Latest Released or Unreleased Procedure version not found in database for procedure name= <i>proc_name</i> (ID:001012) |
| | -20005 | Procedure version not found in database for procedure name= <i>proc_name major_version minor_version</i> (ID:001013) |
| SW_SUSPEND | -20000 | Suspend type (<i>suspend_type</i>) is invalid, expected 2 (suspend) or 0 (activate) (ID:001014) |
| | -20000 | Case (<i>case_num</i>) is already active (ID:001036) |
| | -20000 | Case (<i>case_num</i>) is dead (ID:001037) |
| | -20000 | Case (<i>case_num</i>) is already suspended (ID:001038) |
| SW_GET_SUBPROC_DETAILS | -20000 | Sub-Proc casenum not found in database for Procedure <i>proc_name</i> , Case Number <i>case_num</i> , Step Name <i>step_name</i> , Sub-proc <i>sub_proc_name</i> (ID:001018) |

| Stored Procedure | Oracle Error | Error Text |
|---|--------------|---|
| SW_GETCASE_STATUS | -20006 | Failed to find case information for case: <i>case_num</i> (ID:001019) |
| SW_UNPACKDELAYRELEASEID | -20000 | Unexpected inconsistency between node name <i>nodename</i> and delayed release id procedure node name <i>nodename</i> (ID:001020) |
| | -20000 | Unexpected inconsistency between node name <i>nodename</i> and delayed release id queue node name <i>nodename</i> (ID:001021) |
| SW_PURGE | -20006 | Case (<i>case_num</i>) is dead (ID:001037) |
| | -20006 | Procedure and case information does not match (ID:001041) |
| SW_CLOSE | -20006 | Case (<i>case_num</i>) is dead (ID:001037) |
| | -20006 | 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
# table ssolitemboxtable and the ssolitemboxqueue AQ first.)
#
```

```

swadm add_queue SSOLITEQ1 Local 0001::ssolitemboxtable:ssolitemboxqueue
# 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 Mbox set).
#
swadm add_queue_to_mboxset 6 7
# 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_attribute1 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      in varchar2(31),
    field_value     in varchar2(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.

```
begin
SSOLITE_DATA.SW_ADD_PACK_DATA ('F1', 'DataItem1');
SSOLITE_DATA.SW_ADD_PACK_DATA ('F2', 'DataItem2');
SSOLITE.SW_CASESTART ('CUSTREQ', -1, -1, 'Case1', 'user35', '', casenum, reqid);
SSOLITE.SW_CASESTART ('CUSTREQ', -1, -1, 'Case2', 'user35', '', casenum, reqid);
end;
/
commit;
```

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.

```
begin
SSOLITE_DATA.SW_ADD_PACK_DATA ('F1', 'DataItem1');
SSOLITE_DATA.SW_ADD_PACK_DATA ('F2', 'DataItem2');
SSOLITE.SW_CASESTART ('CUSTREQ', -1, -1, 'Case1', 'user35', '', casenum, reqid);
SSOLITE_DATA.SW_ADD_PACK_DATA ('F1', 'DataItem1');
SSOLITE_DATA.SW_ADD_PACK_DATA ('F2', 'DataItem2');
SSOLITE.SW_CASESTART ('CUSTREQ', -1, -1, 'Case2', 'user35', '', casenum, reqid);
end;
/
commit;
```

SW_ADD_PACK_MEMO

The `SW_ADD_PACK_MEMO` procedure defines an item of pack memo data (a field name/value pair) that will be passed to iProcess with the next command procedure that is called.

Syntax

```
SW_ADD_PACK_MEMO (
    memo_name          in varchar2(31),
    memo_length        in integer,
    memo_data          in long raw,
    array_idx          in integer default 0)
```

where:

- `memo_name` is the name of the iProcess memo field (or memo array field).
- `memo_length` is the number of bytes contained in the memo data.
- `memo_data` is a raw data field that holds the actual memo data.
- `array_idx` (optional) can be specified if `memo_name` is a memo array field; it identifies the specific element in the memo array to be used. If `array_idx` is not explicitly set, it defaults to a value of 0.

If `memo_name` is not a memo array field, you should either not set `array_idx`, or set it to 0. (If `array_idx` contains any other value, no memo data will be found; an error message will be written to the `SWDIR\logs\sw_warn` file.)

Notes `SW_ADD_PACK_MEMO` allows pack memo data to be passed to iProcess when a command procedure is called:

- You must call `SW_ADD_PACK_MEMO` to specify the pack memo data immediately before calling the desired command procedure.
- A call to `SW_ADD_PACK_MEMO` defines a single item of pack memo data. If you 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.

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          in varchar2(8) ,
    proc_maj_ver       in number(5) ,
    proc_min_ver       in number(5) ,
    case_number        in integer ,
    reason             in varchar(2) ,
    user_id            in varchar2(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.

```
EXEC ssolite_data.SW_ADD_PACK_DATA('TEXT1', 'New customer name');
EXEC ssolite.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 (
    proc_name           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    case_num            in integer,
    Audit_id            in integer,
    Audit_step          in varchar2(8),
    Audit_desc          in varchar2(24),
    User_id             in varchar2(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.

```
EXEC SSOLITE.SW_AUDIT('CARPOOL', -1, -1, 51, 131, 'REFUSED', 'BW Activity', 'BW  
User');
```

SW_CASEREOOPEN

The SW_CASEREOOPEN procedure resurrects a case.

Syntax

| | |
|------------------|------------------|
| SW_CASEREOOPEN (| |
| <i>proc_name</i> | in varchar2(8), |
| <i>user_id</i> | in varchar2(24), |
| <i>step_name</i> | in varchar2(8), |
| <i>case_num</i> | in integer) |

where:

- *proc_name* is the name of the procedure that you want to resurrect.
- *user_id* is the name of the iProcess user who is resurrecting the case.
- *step_name* is the name of the case step that you want to resurrect.
- *case_num* is the number of the case that you want to resurrect.

Notes After a case is closed, all the deadlines of the case are removed. If the case is reopened, you can reset the deadlines by running the `CreateCaseDeadline` function. For more information about the `CreateCaseDeadline` function, see *TIBCO iProcess Expressions and Functions Reference Guide*.

Example This example resurrects step STEP1 of case 101 of procedure CUSTREQ.

```
EXEC ssolite.SW_CASEREOOPEN ('CUSTREQ', 'user35', 'STEP1', 101);
```

SW_CASESTART

The SW_CASESTART procedure starts a case of a procedure.

Syntax

```
SW_CASESTART (
    proc_name          in varchar2(8),
    proc_maj_ver       in number(5),
    proc_min_ver       in number(5),
    case_desc          in varchar2(24),
    user_id            in varchar2(24),
    step_name          in varchar2(8),
    case_num           out integer,
    request_id         out integer)
```

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,

- 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.

```
declare
casenum numeric(20);
reqid numeric(20);
begin
SSOLITE_DATA.SW_ADD_PACK_DATA ('CustName', 'Allsop, J.A. ');
SSOLITE_DATA.SW_ADD_PACK_DATA ('CustID', '478163');
SSOLITE.SW_CASESTART ('CUSTREQ', -1, -1, 'Refund request', 'user35', '', casenum,
reqid);

end;
/
commit;
```

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           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    case_number         in integer,
    user_id             in varchar2(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.

```
EXEC SSOLITE.SW_CLOSE ('CUSTREQ', -1, -1, 103, 'swadmin')
```

SW_CLOSE_WITHOUT_EVENT

The `SW_CLOSE_WITHOUT_EVENT` procedure closes an active case of a procedure without triggering the events that are set for the `OnBeforeClose` event or the `OnAfterClose` event.

Syntax

```
SW_CLOSE_WITHOUT_EVENT (
    proc_name           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    case_number         in integer,
    user_id             in varchar2(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 without triggering an event.

```
EXEC SSOLITE.SW_CLOSE ('CUSTREQ', -1, -1, 103, 'swadmin')
```

SW_EVENT

The SW_EVENT procedure triggers a specific event on a case of a procedure.

Syntax

```
SW_EVENT (
    proc_name           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    step_name           in varchar2(8),
    case_num            in integer,
    user_id             in varchar2(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.

```
EXEC SSOLITE.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           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    step_name           in varchar2(8),
    case_num            in integer,
    user_id             in varchar2(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.

```
EXEC SSOLITE.SW_EVENT_UPDATE_PACK ('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 (
    case_num           in integer,
    case_status        out varchar(10),
    proc_type          out varchar(10),
    case_started       out date)
```

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 that the case was started.

Example This example displays the status of case 8.

```
Declare
case_status varchar(10);
proc_type varchar(10);
case_started date;
begin
    SSOLITE.SW_GETCASE_STATUS (8, case_status, proc_type, case_started);
    DBMS_OUTPUT.PUT_LINE('Case status is ' || case_status);
    DBMS_OUTPUT.PUT_LINE('Proc type is ' || proc_type);
    DBMS_OUTPUT.PUT_LINE('Case started at ' || case_started);
end;
/
```

This results in output displaying the status of the case. For example:

```
Case status is Active
Proc type is Sub
Case started at 31-MAY-05
```

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           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    case_number         in integer,
    graft_step_name     in varchar2(8),
    graft_proc_name     in varchar2(8),
    graft_proc_maj_ver  in number(5),
    graft_proc_min_ver  in number(5),
    graft_id            in varchar(2))
```

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.

```
EXEC SSOLITE.SW_GRAFT ('CUSTREQ', -1, -1, 101, 'GRAFT01', 'SUBPROC1', -1, -1,
'UNIQUEID')
EXEC SSOLITE.SW_GRAFTCOUNT ('CUSTREQ', -1, -1, 101, 'GRAFT01', 'UNIQUEID', 1)
/
commit;
```

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 (
    proc_name           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    case_number         in integer,
    graft_step_name     in varchar2(8),
    graft_id            in varchar2,
    graft_count         in 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           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    jump_step           in varchar2(8),
    case_number         in integer,
    jump_reason         in varchar(2),
    user_id             in varchar2(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”.

```
EXEC SSOLITE.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          in varchar2(8),
    tgt_proc_maj_ver      in number(5),
    tgt_proc_min_ver     in number(5),
    src_step_name         in varchar2(8),
    tgt_step_name         in varchar2,
    case_number           in integer,
    jump_reason           in varchar(2),
    user_id               in varchar2(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”.

```
EXEC SSOLITE.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           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    case_number         in integer)
```

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.

```
EXEC SSOLITE.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           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    case_number         in integer)
```

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 (
    proc_name           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    case_number         in integer,
    user_id             in varchar2(24),
    suspend_type        in 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.

```
EXEC SSOLITE.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 (
    proc_name           in varchar2(8),
    proc_maj_ver        in number(5),
    proc_min_ver        in number(5),
    case_number         in integer,
    user_id             in varchar2(24))
```

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.

```
EXEC SSOLITE.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 determined by the value of the `CNUM_SEQ_CACHE` and `REQID_SEQ_CACHE` process attributes).

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*.



The `SW_ENABLECACHING` procedure is part of the `SSOLITE_MSG` package.

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.



The `SW_DISABLECACHING` procedure is part of the `SSOLITE_MSG` package.


```
# Step 6. Start the bulk cases.
#
declare
casenum numeric(20);
reqid numeric(20);
begin
ssolite_data.SW_ADD_PACK_DATA ('CustName', 'Allsop, J.A. ');
ssolite_data.SW_ADD_PACK_DATA ('CustID', '478163');
ssolite.SW_CASESTART ('CUSTREQ', -1, -1, 'Refund request', 'user35', '', casenum,
reqid);

end;
/
commit;

# Step 7. Restore using the default Mbox set.
#
EXEC ssolite.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 in integer)`

where:

- `message_priority` is the priority value.

You can set priorities ranging from 1 to 999, where 1 is the highest priority, for internal message queues when passing messages between iProcess processes such as from the Background process to WIS processes, or from SSOLite to the Background process. Its default value is 50. See [Prioritizing Messages on page 262](#) for more information.

Notes Use the `SW_UNSET_PRIORITY` procedure to restore the default message queue priorities.

Example The following example sets the `SW_CASESTART` priority of Case1 and Case2 to 70, Case3 and Case4 to 100, and Case5 to the default priority.

```
begin
ssolite.SW_SET_PRIORITY (70);
ssolite.SW_CASESTART ('CUSTREQ', -1, -1, 'Case1', 'user35', '', 0, 0);
ssolite.SW_CASESTART ('CUSTREQ', -1, -1, 'Case2', 'user35', '', 0, 0);
ssolite.SW_SET_PRIORITY (100);
ssolite.SW_CASESTART ('CUSTREQ', -1, -1, 'Case3', 'user35', '', 0, 0);
ssolite.SW_CASESTART ('CUSTREQ', -1, -1, 'Case4', 'user35', '', 0, 0);
ssolite.SW_UNSET_PRIORITY ();
ssolite.SW_CASESTART ('CUSTREQ', -1, -1, 'Case5', 'user35', '', 0, 0);
end;
/
commit;
```

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 261](#) 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 262](#) 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:

```
TEMPORARY TABLE SSOLITE_DEBUG_DATA (  
    message          varchar(256));
```

The table simply holds the debug message text in inserted order. If an application has enabled debugging, a simple `select * from 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 in integer)`

where *enable* is a flag that turns debugging on or off. Specify:

- 1 to enable debugging (and create the `SSOLITE_DEBUG_DATA` temporary table).
- 0 to disable debugging. Note that any existing debug information written to the `SSOLITE_DEBUG_DATA` temporary table for the current transaction is still available.

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 You must call SW_GET_DEBUG to populate the SSOLITE_DEBUG_DATA table. Calling the procedure copies data from the Oracle DBMS_OUTPUT package to SSOLite's own data type.

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 D **Database Stored Procedures**

This appendix describes the package of database stored procedures.

Topics

- [Overview, page 310](#)
- [CASENUM_FIND_GAPS, page 311](#)

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 37](#) 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.

```
conn swpro_11309/swpro_11309@ORCL;
set serveroutput on
BEGIN
    CASENUM_FIND_GAPS ( 100, 500, 20);
END;
/
quit;
```

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 E Unused Tables

The following tables are created by the database creation script (`init2Kora.sql` on UNIX, `init2Kora_tok.sql` on Windows), 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.

