TIBCO iProcess® Server Objects (Java)

Programmer'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, TIBCO iProcess, and Two-Second Advantage 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 © 2002-2014 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Table of Contents

Preface . . . . . **xii**
Introduction xii
Product Name Changes xii
Knowledge Level xiii
Documentation Set xiii

Revision History . . . . . **xiv**

Chapter 1 — Introduction **1**
Introduction 1
Procedures 1
TIBCO iProcess Server Objects Design 3
Available Interfaces 3
TIBCO Process / iProcess Engine 4
Engine and Server Version Numbers 4
SWDIR - The System Directory 5
TIBCO iProcess Objects Server 5
TIBCO iProcess Objects Director 5

Chapter 2 — Naming Conventions **6**
Naming Conventions Used in TIBCO iProcess Server Objects (Java) 6

Chapter 3 — Object Types **8**
Object Types 8
Server Objects 10
Garbage Collection for Server Objects 11
What is a User Session? 11
Disconnecting User Sessions 12
Sharing User Sessions 12
Server Object Parameters 13
Delegate Objects 13
Value Objects 14
Serialization 14
Multiple Value Objects 14
Constructing Value Objects 15
Are Objects Reentrant? 15

Chapter 4 — Interfaces **16**
Introduction 16
JBase Implementation 16
Process Flow 17
Packages 17
JAR File 17
RMI Implementation 17
Process Flow 18
Packages 19

JAR File . . . 19
RMI Components . . . 19
Remote Interface . . . 19
Remote Interface Implementation . . . 19
Server Factory . . . 20
Command-Line Options . . . 20
Registry Location . . . 22
Naming Factory . . . 22
Delegate Objects . . . 23
Application Servers . . . 23
External Rebind Needed When Using Either WebSphere JAAS or JBoss . . . 24
External Rebind for WebSphere JAAS . . . 24
External Rebind for JBoss . . . 26
EJB Implementation . . . 27
Custom EJB Solution . . . 27
Process Flow . . . 27
EJB Solution . . . 28
Process Flow . . . 28
Packages . . . 29
EJB Components . . . 29
EJBHome Remote Interface . . . 30
EJBObject Remote Interface . . . 30
Enterprise Bean Implementation . . . 30
Delegate Objects . . . 30

Chapter 5 — Node Management . . . 32
Introduction . . . 32
Constructing a vNodeId Object . . . 33
Sending a Directed UDP Message . . . 34
Specifying a UDP Port . . . 34
Multiple Instances of the TIBCO iProcess Objects Server / Director . . . 35
Sending a UDP Broadcast . . . 35
Setting the UDP Broadcast Interval . . . 35
Specifying a UDP Port . . . 36
Multiple Instances of the TIBCO iProcess Objects Server / Director . . . 36
What if a Known Node is not Answering the UDP Broadcast? . . . 37
Configuring the TIBCO iProcess Objects Server TCP Port . . . 38
Configuring the TCP Port on a Windows System . . . 38
Configuring the TCP Port on a UNIX System . . . 39
Using TIBCO iProcess Server Objects Through a Firewall? . . . 40
Database Configuration . . . 41
Database Configuration Access . . . 41
Activity Publication . . . 42
Activity Publication Access . . . 42
Configuring Activity Publication . . . 42
Using the sIPEConfig Object . . . 43
Configuration Example . . . 43

Chapter 6 — Procedures . . . 46
Introduction . . . 46
Managing Procedures . . . 46

Procedure Version Control 48
Procedure Status 48
Accessing the Procedure Version Number 49
Procedure Version Number in the Audit Trail 49
Procedure Version Details 50
Listing Versions of a Procedure 50
Accessing a Specific Procedure Version 50
Procedure Audit Trails 51
Sub-Procedures 52
Sub-Procedure Call Steps 52
Accessing Sub-Procedure Call Step Definitions 53
Sub-Procedure Start Precedence 53
Dynamic Sub-Procedure Call Steps 53
Accessing Dynamic Sub-Procedure Call Step Definitions 56
Passing Data between a Main and Sub-Procedure 56
The Sub-Case Object 56
SubProcPath to a Sub-Case 57
Public Steps 59
Public Fields 60
Retrieving Public Field Objects 60

Chapter 7 — Case Management 61
Starting a Case 61
Case Description 62
Keeping/Releasing the Start Step 62
Validating Markings on the Start Step (iProcess Modeler Forms Only) 63
Sub-Procedure Precedence 64
Why isn't the Started Case Appearing in the Work Queue? 65
Obtaining the Case Number of a Case that was just Started 65
Determining Who Can Start a Case 66
Which Procedures can a User Start? 66
Obtaining Lists of Cases 67
Determining the Number of Cases in a Procedure 68
Auditing Cases 69
Determining who can Audit Cases of a Procedure 69
Which Procedures can a User Audit? 70
Audit Step Objects 70
Getting Audit Step Objects 72
Configuring Audit Trail Strings 73
Auditing Sub-Procedures 74
Grouping Sub-Cases in the Audit Trail 74
Filtering Audit Data 75
Creating an Audit Filter Expression 75
Adding User-defined Audit Trail Entries 80
Custom Audit Trail Message Templates 81
Withdraw Outstanding Items / Jump To New Steps 81
Determining Outstanding Items 82
ProcPath to Outstanding Items 83
Triggering Events 85
Predicting Cases 85

Defining Case Prediction . . . 86
Step Duration. . . . 86
Conditional Actions for Case Predictions . . . 87
Performing Case Prediction . . . 87
Background Case Prediction . . . 87
Live Case Prediction . . . 88
Case Simulation. . . . 88
Sub-Procedures, Dynamic Sub-Procedures, and Graft Steps in Prediction . . . 89
Sub-Procedure Call Steps . . . 89
Dynamic Sub-Procedure Call Steps and Graft Steps . . . 89
Including Case Data Queue Parameter Data in Prediction Results . . . 89
Filtering and Sorting Predicted Items. . . . 90
Using Graft Steps . . . 91
Defining Graft Steps. . . . 92
Starting a Graft Task. . . . 92
Setting the Task Count . . . 93
Outstanding Graft Items . . . 94
Return Statuses . . . 95
Deleting a Task . . . 95
Completing a Graft Step. . . . 96
Error Processing . . . 96
Transaction Control Steps. . . . 98
The vTransactionControlStep Object . . . 98
Type of Transaction Control Step. . . . 98
Outstanding Transaction Control Steps . . . 99
Retrying Failed Transactions . . . 99
Audit Trail Messages . . . 100
Suspending Cases . . . 100
Reactivating a Suspended Case . . . 101
Ignoring Case Suspension . . . 101
Closing Cases . . . 102
Resurrecting a Closed Case . . . 102
Purging Cases . . . 102

Chapter 8 — Managing Work Queues . . . 104
Introduction. . . . 104
Work Queue Objects. . . . 104
Work Item Objects . . . 104
Test vs. Released Work Queues . . . 104
Retrieving Work Queues. . . . 106
Filtering Content when Retrieving Work Queues . . . 107
Retrieving Work Items . . . 108
Filtering and Sorting Work Items . . . 109
Filtering Content when Accessing Work Items . . . 109
Work Queue Deltas. . . . 110
Work Queue Deltas With Pageable Lists . . . 110
Delta Status . . . 112
Work Queue Deltas Via a JMS Topic. . . . 112
JMS Deltas When Using Pageable Lists — JBase or RMI Interface . . . 113
JMS Deltas When Using Single-Block Item Access — JBase or RMI Interface. . . . 115

JMS Deltas When Using Single-Block Item Access — XML Interface . . . 117
Work Queue Delta JMS Messages . . . 119
Processing Work Items . . . 121
Locking Work Items . . . 121
Controlling Fields Returned when Locking Work Items . . . 121
What's the Difference Between a "Lock" and a "Long Lock"? . . . 122
Locking the First Available Work Item in a List . . . 122
Pageable Lists . . . 123
Single-Block Item Access Lists . . . 123
Unlocking a Work Item . . . 123
Discarding Changes made to a Locked Work Item . . . 124
Has a Work Item been Locked/Opened? . . . 124
Determining who Locked a Work Item . . . 124
Executing a Command when a Work Item is Locked . . . 124
Keeping Work Items . . . 124
Executing a Command when a Work Item is Kept . . . 125
Releasing Work Items . . . 125
Validating Markings . . . 125
Executing a Command when a Work Item is Released . . . 126
Automatically Releasing the Start Step . . . 126
What is an Orphaned Work Item? . . . 126
Determining if a Work Item could not be Delivered to the Addressee . . . 126
Is the Work Item Directly Releasable? . . . 126
Errors Resulting from Processing Work Items . . . 127
Work Item Deadlines . . . 127
Deadline Withdrawal . . . 127
Filtering and Sorting on Deadline Information . . . 128
Dynamically Recalculating Deadlines . . . 128
Keeping a Work Item that is Withdrawn . . . 129
Participating in Another User's Work Queue . . . 129
Participation Schedules . . . 130
Modifying Existing Participation Schedules . . . 130
Creating Participation Schedules . . . 130
Removing Participation Schedules . . . 130
Using the vDate and vTime Objects in Participation Schedules . . . 131
Forwarding/Redirecting Work Items to Another Work Queue . . . 131
Manually Forwarding Work Items . . . 131
Determining the Work Queues to which a Work Item can be Forwarded . . . 132
Automatic Forwarding/Redirecting Work Items . . . 133
Redirection Schedules . . . 133
Modifying an Existing Redirection Schedule . . . 133
Cancelling a Redirection Schedule . . . 133
Using the vDateTime Object in Redirection Schedules . . . 134
Work Queue Supervisors . . . 134
Adding Work Queue Supervisors . . . 134
Removing Work Queue Supervisors . . . 135
External Work Items . . . 135
Releasing an External Work Item . . . 135

Chapter 9 — Working with Lists 137
Introduction 137
Using Single-Block Item Access 137
Making a List 138
make<type>List Method Input Parameters 138
From the XML Interface: 138
From the JBase, RMI, and SI Interfaces: 139
make<type>List Method Return Values 140
From the XML Interface: 140
From the JBase, RMI, and SI Interfaces: 140
Fetching a List 141
fetch<type>List Method Input Parameters 141
From the XML Interface: 141
From the JBase, RMI, and SI Interfaces: 141
fetch<type>List Method Return Values 143
From the JBase, RMI, and SI Interfaces: 143
Fetch “If Changed” Methods 143
List State Objects 144
Base Object 144
Filter and Sort Criteria Information 145
Content Information 145
Summary Information 146
Work Item Status Information 147
Requested Items 147
Using Pageable Lists 148
Using Pageable Lists with Work Items 149
Refreshing a Pageable List of Work Items 151
Refresh Status 151
Using a Director or Multiple Instances of the TIBCO iProcess Objects Server 152
Using Pageable Lists with Cases, WorkQs, Groups, Users, and OSUsers 153
Held Pageable Lists 155
The isReturnAllFields Flag is always False on Held Pageable Lists 156
Access Permissions 156
Pageable List Counts 156
Controlling Pageable List Resources 157
Client Resources 158
Setting the Size of the Blocks 159
Automatically Clearing Blocks 159
Explicitly Clearing Blocks 159

Chapter 10 — Retrieving Dependent Objects 160
Introduction 160
Content Request Objects 161
Using Content Request Objects 162
Retrieving Dependent Objects 162
Controlling Case Data 163
Controlling Fields Returned when Locking Work Items 163

Chapter 11 — iProcess Fields 165
What is an iProcess Field? 165
Case Data vs. Work Item Data 166

Including Field Data when Starting a Case . . . 166
Setting Case Data . . . 167
Uninitialized Fields . . . 167
Parallel Steps . . . 167
Retrieving Field Data from the Server . . . 167
Including Field Data when Retrieving Cases . . . 168
Including Field Data when Retrieving Work Items . . . 168
Including Field Data when Locking Work Items . . . 168
Case Data Queue Parameter Fields . . . 169
Passing Field Data when Keeping/Releasing Work Items . . . 169
What are Markings? . . . 170
Type Validation on Fields/Markings . . . 170
Accessing Memo Fields . . . 171
Accessing Attachments . . . 171
Accessing System Fields . . . 171
Array Fields . . . 172
Array Field Indexes . . . 173
Using Array Fields in Filter Expressions . . . 174
Requesting, Returning, and Setting All Array Field Elements . . . 174
Requesting All Array Field Elements . . . 174
Returned Array Field Elements . . . 175
SWFieldType Enumerations Used With Array Fields . . . 176
Setting All Array Field Elements . . . 176
Example . . . 176
Setting Array Field Values to SWEmptyField in XML . . . 177
Date Format . . . 178
Character Encoding . . . 179

Chapter 12 — User Administration . . . 180
Introduction . . . 180
Types of Users . . . 180
MOVESYSINFO Function . . . 181
iProcess Users . . . 182
Creating an iProcess User . . . 182
Deleting an iProcess User . . . 183
Is an O/S User needed for every iProcess User? . . . 183
Changing the User's Password . . . 183
User Groups . . . 184
Creating a User Group . . . 184
Deleting a User Group . . . 185
Adding and Removing Users to/from a Group . . . 185
Roles . . . 186
Creating a Role . . . 186
Deleting a Role . . . 186
User Attributes . . . 187
Modifying an Existing Attribute Value . . . 189
Creating an Attribute Definition . . . 189
Deleting an Attribute . . . 189
Why isn't the new User, Group, Role or Attribute Available? . . . 190
User Authority . . . 190

User Preference Data 192

Chapter 13 — Filtering Work Items and Cases (No Enhancements) 193
Introduction 194
Specifying Filter Criteria 194
Defining Filter Expressions 195
Number of Cases or Work Items in a Filtered Pageable List 196
Filtering/Sorting in an Efficient Manner 197
Filtering/Sorting Work Items 197
Getting Case Data 199
Can the WIS Perform the Filter Operation? 199
Work Item Server vs. TIBCO iProcess Objects Server Example 201
Can the WIS Perform the Sort Operation? 201
Filtering/Sorting Cases 202
Getting Case Data 204
Filtering Cases on the TIBCO iProcess Objects Server 204
Efficiently Filtering Cases on the TIBCO iProcess Objects Server 205
Sorting Cases on the TIBCO iProcess Objects Server 205
Getting Audit Data 206
Filter Criteria Format 206
System Fields Used in Filtering 208
Data Types used in Filter Criteria 211
Data Type Conversions 211
Filtering on Case Data Fields 212
Using Case Data Queue Parameter Fields 212
Type of Data in CDQPs 212
Using Work Queue Parameter Fields 213
Work Queue Parameter Fields vs. Case Data Queue Parameter Fields 214
Using Regular Expressions 215
Using Escape Characters in the Filter Expression 216
Filtering on Empty Fields 216
How to Specify Ranges of Values 217
Specifying Multiple Ranges 217
Closing/Purging Cases Based on Filter Criteria 218
Default Filter Criteria 219

Chapter 14 — Filtering Work Items and Cases (with WIS Work Item Filtering) 221
Introduction 222
Specifying Filter Criteria 222
Defining Filter Expressions 223
Number of Cases or Work Items in a Filtered Pageable List 224
Filtering/Sorting in an Efficient Manner 225
Filtering/Sorting Work Items 225
Getting Case Data 227
Work Items are Filtered by the WIS 227
Can the WIS Perform the Sort Operation? 228
Filtering/Sorting Cases 228
Getting Case Data 230
The TIBCO iProcess Objects Server Filters Cases 230
Efficiently Filtering Cases on the TIBCO iProcess Objects Server 231
The TIBCO iProcess Objects Server Sorts Cases 231

Getting Audit Data . . . 232
Filter Criteria Format . . . 232
System Fields Used in Filtering . . . 234
Data Types used in Filter Criteria . . . 236
Data Type Conversions . . . 237
Filtering Work Items on the WIS . . . 237
Filtering Cases on the TIBCO iProcess Objects Server . . . 237
Filtering on Case Data Fields . . . 238
Using Case Data Queue Parameter Fields . . . 238
CDQPs Contain Work Item Data . . . 239
Using Work Queue Parameter Fields . . . 239
Work Queue Parameter Fields vs. Case Data Queue Parameter Fields . . . 240
Using Regular Expressions . . . 241
Regular Expressions with Work Item Filtering . . . 241
Regular Expressions with Case Filtering . . . 242
Using Escape Characters in the Filter Expression . . . 243
Filtering on Empty Fields . . . 244
How to Specify Ranges of Values . . . 244
Closing/Purging Cases Based on Filter Criteria . . . 245
Default Filter Criteria . . . 245

Chapter 15 — Filtering Work Items and Cases (with WIS Work Item and Database Case Filtering) 247
Introduction . . . 248
Specifying Filter Criteria . . . 248
Defining Filter Expressions . . . 249
Length of Filter Expressions . . . 250
Large Filter Expressions May Require Larger Stack Size . . . 250
Number of Cases or Work Items in a Filtered Pageable List . . . 250
Filtering/Sorting in an Efficient Manner . . . 251
Filtering/Sorting Work Items . . . 252
Getting Case Data . . . 253
Work Items are Filtered by the WIS . . . 253
Can the WIS Perform the Sort Operation? . . . 254
Filtering/Sorting Cases . . . 255
Getting Case Data . . . 256
The Database Filters Cases . . . 257
The Database Sorts Cases . . . 258
Getting Audit Data . . . 258
Filter Criteria Format . . . 258
System Fields Used in Filtering . . . 260
Data Types used in Filter Criteria . . . 263
Data Type Conversions . . . 263
Filtering on Case Data Fields . . . 264
Using Case Data Queue Parameter Fields . . . 264
CDQPs Contain Work Item Data . . . 265
Using Work Queue Parameter Fields . . . 265
Work Queue Parameter Fields vs. Case Data Queue Parameter Fields . . . 266
Using Regular Expressions . . . 267
Regular Expressions with Work Item Filtering . . . 267
Regular Expressions with Case Filtering . . . 268

Using Escape Characters in the Filter Expression . . . 269
Filtering on Empty Fields . . . 269
How to Specify Ranges of Values . . . 269
Closing/Purging Cases Based on Filter Criteria . . . 270
Default Filter Criteria . . . 270

Chapter 16 — Sorting Work Items and Cases . . . 273
Introduction . . . 273
Specifying Sort Criteria . . . 273
Sorting in an Efficient Manner . . . 274
System Fields used in Sorting . . . 275
Sorting on Case Data Fields . . . 277
Using Case Data Queue Parameter Fields . . . 277
Using Work Queue Parameter Fields . . . 278
Setting Default Sort Criteria . . . 279
Sorting as a Specified Data Type . . . 280

Chapter 17 — Error Handling . . . 281
Introduction . . . 281

Chapter 18 — Client Configuration . . . 283
Client Log . . . 283
Client Log Overview . . . 283
Controlling the Client Log . . . 285
The sClientLog Object . . . 285
Registry Settings (Windows only) . . . 285
Environment Variables (UNIX Only) . . . 286
Name and Location of the Client Log . . . 287
Log File Name . . . 287
Log File Directory . . . 287
Construction of Client Log Name and Directory . . . 287
Activating / Deactivating the Client Log . . . 288
Filtering the Client Log . . . 288
Setting the Log Level . . . 288
Filtering by Category . . . 289
Enabling and Disabling Categories . . . 289
Filtering by Message . . . 290
Enabling and Disabling Messages . . . 290
Adding Entries to the Client Log . . . 291
Setting the Size of the Client Log . . . 291
Resetting the Client Log . . . 291
Message Wait Time . . . 292
Character Encoding Using ICU Conversion Libraries . . . 293

Chapter 19 — XML Interface . . . 294
Introduction . . . 294
XML Server Objects . . . 294
Using the XML Interface . . . 295
Constructing the xSession Object . . . 295
XML Results . . . 296

Example XML Data 297
Returning Lists of Items 299
Dates and Times 299
Error Handling 300
Object Serialization 303
Serialize/Deserialize Functions 304

Index 305

Preface

Introduction

The *TIBCO iProcess Server Objects (Java) Programmer's Guide* describes the technical aspects of the TIBCO iProcess Server Objects (Java), a set of programming objects used to build applications that automate business processes.

This document provides descriptions of the functionality of the TIBCO iProcess Server Objects (Java). For details about the syntax for the methods used to provide that functionality, see the on-line help provided with the product.

The Revision History page (see page xiv) shows the version number of TIBCO iProcess Server Objects (Java) for each issue of this document. If you are using an older version of TIBCO iProcess Server Objects (Java), you may experience functionality that is different than what is described in this document.

For the latest product information, please refer to the TIBCO Support Services website at http://www.tibco.com/services/support.

Product Name Changes

Staffware, the original producer of this product, was purchased by TIBCO Software Inc. in 2004. As a result of this purchase, product names have undergone a change. The table below shows how product names have changed from Staffware to TIBCO.

Staffware Name	TIBCO Name
Staffware Process Objects (SPO) Server	TIBCO iProcess® Objects Server
Staffware Process Objects (SPO) Director	TIBCO iProcess® Objects Director
Staffware Server Objects (SSO) for .NET	TIBCO iProcess® Server Objects (.NET)
Staffware Server Objects (SSO) for EJB	TIBCO iProcess® Server Objects (Java)
Staffware iProcess Engine	TIBCO iProcess® Engine
Staffware Process Definer (SPD)	TIBCO iProcess® Modeler

Although the name "Staffware" has been removed from the product names, these products are all part of a suite of products called the "TIBCO Staffware Process Suite."

You may still see references to Staffware, SPO, and SSO in the software (e.g., file and directory names) and documentation.

Also note that some products that were referred to as “clients” have recently undergone a name change, as follows:

Previous Name	New Name
TIBCO iProcess® Client (Windows)	TIBCO iProcess® Workspace (Windows)
TIBCO iProcess® Client (Browser)	TIBCO iProcess™ Workspace (Browser)®

Knowledge Level

The intended audience of this document is programmers and technical consultants who are using the objects that comprise TIBCO iProcess Server Objects (Java) to create client applications for TIBCO customers or the customers of TIBCO partners.

Documentation Set

In addition to this document, the following make up the documentation set for this product:

- *TIBCO iProcess Server Objects (Java) On-Line Help* - This provides syntax information for all objects and methods available to programmers. This is an HTML-based help system that can be started by pointing your browser to *InstallDir***Help\start.htm** (Windows) or *InstallDir*/**JavaHelp/start.htm** (UNIX)
- *TIBCO iProcess Server Objects (Java) Object Model Graphic* - This provides a graphical representation of the objects that comprise the TIBCO iProcess Server Objects (Java).
- *TIBCO iProcess Server Objects (Java) Installation* - This guide provides the steps you need to follow to successfully install your TIBCO iProcess Server Objects (Java) software.
- *TIBCO iProcess Server Objects (Java) Release Notes* - The release notes provide information about changes that have occurred in each release of TIBCO iProcess Server Objects (Java). It may also include information about last-minute changes that are not included in the help system or programmer's guide.
- *TIBCO iProcess Objects Server Administrator’s Guide* - This provides information about starting, stopping, and configuring the TIBCO iProcess Objects Server, which is used in conjunction with the TIBCO iProcess Server Objects (Java). It also includes information about how to set up the TIBCO iProcess Objects Server log.
- *TIBCO iProcess Objects Director Administrator’s Guide* - This provides information about how to use and configure the TIBCO iProcess Objects Director, which can be used to manage connections between your client application and TIBCO iProcess Objects Servers.

Revision History

Issue	Date	Current Version of TIBCO iProcess Server Objects (Java)	Summary of Changes
Issue 1	Nov. 2002	9.0(0.0)	Initial release
Issue 2	Dec. 2002	9.0(0.1)	Misc. changes throughout
Issue 3	June 2003	9.0(2.0)	Misc. changes throughout
Issue 4	Dec. 2003	10.0(2.0)	Added *Procedures* chapter. Added information about predicting cases, jumping to new outstanding steps, using graft steps, and using dynamic sub-procedures. The *Filtering Work Items and Cases* chapter was split into three separate chapters. The reader will use the appropriate chapter, depending on the filter enhancements that have been implemented in their TIBCO iProcess Objects Server.
n/a	August 2005	10.2.3	Added information about using the TIBCO iProcess Objects Director, running multiple instances of the TIBCO iProcess Objects Server on Windows (configuration utility changes), moveSysInfo method, transaction control steps, new vDate, vTime, and vDateTime objects used when creating participation and redirection schedules, etc. Also note that the issue number was removed from the title page to reflect the standard used by TIBCO — the title page now contains the month and year the document was issued, as well as the version number of TIBCO iProcess Server Objects at time of issue.
n/a	Oct. 2005	10.3.0	Added information about activity publication. Removed information about the TIBCO iProcess Objects Server (starting/stopping, configuration parameters, logging, etc.) — for this information, see the *TIBCO iProcess Objects Server Administrator's Guide*. Also removed information about the TIBCO iProcess Objects Director — for this information, see the *TIBCO iProcess Objects Director Administrator's Guide.*
n/a	Feb. 2006	10.3.1	Added *XML Interface* chapter to describe using the XML interface. Also changed the name of the *Working with Pageable Lists* chapter to *Working with Lists*, and added information about using the new "single-block item access" lists.
n/a	July 2006	10.5.0	Added information about dynamically recalculating deadlines and work queue deltas.
n/a	May 2007	10.6.0	Removed references to swadmin and swpro (changed to IPEADMIN user).
n/a	December 2007	10.6.1	No functional changes to iProcess Server Objects (Java). Minor corrections in this document.

Revision History
(Cont.)

Issue	Date	Current Version of TIBCO iProcess Server Objects (Java)	Summary of Changes
n/a	May 2008	11.0.0	Added new “fetch” methods that return work items only if there has been a change in work items on the list since you originally obtained it. Added information about obtaining work queue deltas via a JMS topic. Added information about object model changes to allow you to set, get, and delete user preference data as a text string in the iProcess Engine database.
n/a	Jan. 2009	11.1.0	Added getMaxCnt method to the vACaseContent object, as well as the getOverMaxCnt method to the vSummary object. Added info throughout concerning markings only being applicable to iProcess Modeler-produced forms. Added new table that lists all possible audit trail filter types, including the numeric values and equates that represent each type.
n/a	Feb. 2010	11.2.0	Added information about new “lock first available work item” methods: lockFirstItem, lockFirstWorkItem, and lockFirstAWorkItem.
n/a	Oct. 2010	11.3.0	No new functionality in 11.3.0. Minor corrections.
n/a	April 2012	11.4.0	Minor changes.
n/a	April 2014	11.4.1	Minor changes.

1

Introduction

Introduction

TIBCO iProcess Server Objects (Java) comprise a set of objects that are used to build applications that automate business processes. TIBCO iProcess Server Objects (Java) consists of an object model that provides access to the information and functionality needed in these applications.

The objects in the TIBCO iProcess Server Objects (Java) object model can be used to start cases, present information on screens to users, manipulate work items, remind users when actions need to be taken, and monitor and control the flow through the business process.

Procedures

TIBCO refers to a business process as a "procedure." Procedures are defined with a tool called the TIBCO Business Studio™. A procedure consists of a number of "steps," including manual steps (which require user action), automatic steps (which are executed automatically by the server), and condition steps (which branch based on the result of a condition). An example of a simple procedure is shown below.

Example of a Procedure

Before describing the underlying architecture of TIBCO iProcess Server Objects (Java), it's important to understand the terminology used with procedures. The following table provides definitions of some of the key terms that are used throughout this document.

Term	Definition
Procedure	Represents the definition of a business process, which ensures that information flows in a consistent and timely manner through the system. A procedure is defined using TIBCO Business Studio. An example is shown in the illustration above.

Term	Definition
Case	This is a particular instance of a procedure. A case is created when a procedure is started, and remains in existence until that instance of the procedure is purged from the system.
Step	A procedure is made up of a number of steps, which define the activities that take place within the flow of a procedure. Each step defines what must be done, who must do it, and, optionally, a deadline by which it must be done.
Work Queue	This is a list of work items (see below) that are awaiting action. A work queue can belong to an individual user or to a group of users. If it is a group work queue, any user that belongs to that group has access to the work items in that group queue.
Work Item	A work item represents an action item listed in a work queue. It relates to a step in an active case. A user manages the work items in their work queue by performing some sort of action upon them, such as entering data on a form, forwarding the item to another user or group, “keeping” it (placing it back in the work queue for further action at a later time), or “releasing” it (completing the required action and sending it on to the next step in the procedure).
Node	A node represents a TIBCO iProcess Objects Server. Each node “owns” its own users, groups, procedures, work queues, etc. To the procedures that it owns, the node is known as the “hosting node.”
User	A user is an individual who has been defined on a node (TIBCO iProcess Objects Server), giving that user access privileges to log in to that server. Each user has a work queue with the same name as the user name defined on the node.
Group	A group represents a collection of users. Each group has a work queue that has the same name as the group name defined on the node. All users that are members of the group have access to the group work queue.

All of the properties and functionality associated with these items (cases, steps, etc.) are exposed by TIBCO iProcess Server Objects (Java). Client applications have use of those properties and functionality in the business processes they automate.

TIBCO iProcess Server Objects Design

TIBCO iProcess Server Objects is designed to be used in server-side application architectures. The TIBCO iProcess Server Objects object model has been designed to accommodate these types of applications by incorporating the following features:

- A “flat” object hierarchy - Being able to reach the desired data without having to traverse a “deep” object hierarchy reduces the resources needed and the time required to reach the desired data.
- A very granular interface - Because of the communications overhead of a distributed system, a very granular interface is called for. This results in fewer method calls moving more data per call to the client.
- Efficient data grouping - This prevents the moving of unnecessary data and provides a more responsive system.

Client applications make use of the objects in the TIBCO iProcess Server Objects by making method calls that either retrieve or modify data. These method calls cause messages to be sent to a **TIBCO iProcess Objects Server**. The TIBCO iProcess Objects Server acts as a gateway between the client application created with TIBCO iProcess Server Objects, and the TIBCO iProcess Engine, where the actual processing and storage of data occurs. The TIBCO iProcess Engine manages all data, routing work items and updating the appropriate work queues.

*Note - There are two “types” of engines: the **TIBCO Process Engine** and the **TIBCO iProcess Engine**. The TIBCO iProcess Server Objects object model can be used with either of these engines. However, there are some functionality differences between the engines. These differences are noted in the on-line help system. For more information, see* *page 4*.

Available Interfaces

TIBCO iProcess Server Objects (Java) has one true Java interface — the **SI Interface** (for “Server Interface”). There are two implementations of the SI Interface available:

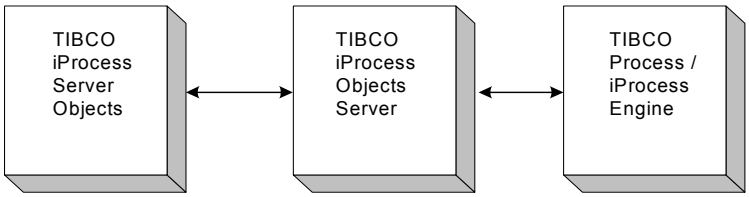
- **JBase** - This is the local solution — it exposes TIBCO iProcess Server Objects functionality as simple Java objects. This interface does not make use of remote objects. It is typically used when incorporating a broker application that is automatically processing work items arriving in a particular work queue.
- **RMI** - This is the remote solution — it uses Java’s Remote Method Invocation (RMI) technology. This allows the client to be located on a machine remote from the TIBCO iProcess Server Objects.

  Note - There is also an Enterprise JavaBeans (EJB) implementation of TIBCO iProcess Server Objects available, which provides the ability to perform business logic functions through EJBs in the middle tier. This architecture, which makes use of a Java application server for scalability, conforms to the J2EE specification. Note, however, that as of version 10.3.0 of the TIBCO iProcess Server Objects (Java), the EJB interface is no longer being updated with new features and enhancements. It is still available, but is being phased out of the product line.

The interfaces described above provide data in the form of objects. There is also an XML interface available that allows you to retrieve all data from the TIBCO iProcess Engine as an XML data stream rather than in the form of objects. For more information about the XML interface, see *XML Interface* on page 294.

TIBCO Process / iProcess Engine

The “engine” manages all TIBCO data, routing work items and updating the appropriate work queues.



There are actually two “types” of engines:

- **TIBCO iProcess Engine -** This type of engine is required for some of the newer functionality of TIBCO iProcess Server Objects. If you are using a TIBCO iProcess Engine, you will also be using a TIBCO iProcess Objects Server that supports the functionality provided by the TIBCO iProcess Engine.
- **TIBCO Process Engine -** If you are using this type of engine, some of the newer functionality of TIBCO iProcess Server Objects is not available to you. If you are using a TIBCO Process Engine, you will also be using a TIBCO iProcess Objects Server that supports the functionality provided by the TIBCO Process Engine.

TIBCO iProcess Server Objects will work with both “types” of engines described above — the difference is the amount of functionality available from the engine. The on-line help system provides information about which functionality is available only from the TIBCO iProcess Engine.

Engine and Server Version Numbers

As we are transitioning from "Staffware" to "TIBCO," the version numbers of the engines and servers are changing as well. Staffware version numbers included major, minor, maintenance release, and patch numbers, with parentheses (e.g., 10.2(0.0)). TIBCO version numbers include major, minor, and maintenance release numbers, without parentheses (e.g., 10.2.0). Hotfix numbers (the equivalent to a "patch") are not shown in the product version number.

A Staffware version number may also be preceded by an "i" (e.g., i10.0(0.0)), indicating that it is an "iProcess" Engine or a TIBCO iProcess Objects Server that supports the functionality offered by iProcess Engines.

Moving forward from version 10.2.0, all new releases of engines, TIBCO iProcess Objects Servers, and TIBCO iProcess Server Objects will use the 3-digit TIBCO version numbering system. The version number will also not include an "i" to indicate that it is an iProcess Engine or a TIBCO iProcess Objects Server that supports the functionality of an iProcess Engine; by default, all engines from 10.2.0 forward are iProcess Engines, and all TIBCO iProcess Objects Servers from 10.2.0 forward support the functionality of iProcess Engines.

You can determine whether you are using a TIBCO Process Engine or a TIBCO iProcess Engine by looking at the version number. The version number can be found in the first line of the *SWDIR***swdefs** (Windows) or $SWDIR/**swdefs** (UNIX) file.

SWDIR - The System Directory

The directory where the TIBCO Process/iProcess Engine is installed is known as the *system directory.* It is referred to in this guide as *SWDIR*.

On UNIX systems, the environment variable $SWDIR should be set up to point to the system directory for the **root** and "system administrator" user.

Note - The "system administrator" user can be designated as any iProcess user when the TIBCO iProcess Engine is installed; it defaults to the user installing the iProcess Engine. This user is known as the "IEPADMIN" user.

TIBCO iProcess Objects Server

The TIBCO iProcess Objects Server acts as a gateway to pass data between the TIBCO iProcess Server Objects and the TIBCO Process/iProcess Engine. Note that only certain versions of the TIBCO iProcess Objects Server can be used with each of the different types of TIBCO Process/iProcess Engines:

- If you are using a TIBCO iProcess Engine, you must use a TIBCO iProcess Objects Server that supports the functionality provided by the TIBCO iProcess Engine.
- If you are using a TIBCO Process Engine, you must use a TIBCO iProcess Objects Server that supports the functionality provided by the TIBCO Process Engine.

For information about starting/stopping and configuring the TIBCO iProcess Objects Server, see the *TIBCO iProcess Objects Server Administrator's Guide*.

TIBCO iProcess Objects Director

The TIBCO iProcess Objects Director is a standalone program that maintains a list of TIBCO iProcess Objects Servers that are configured in a node cluster. When a client application needs access to a TIBCO iProcess Objects Server, it first establishes a connection to the TIBCO iProcess Objects Director. The TIBCO iProcess Objects Director then decides, based on a "pick method," which TIBCO iProcess Objects Server the client should connect to.

The list of known TIBCO iProcess Objects Servers is updated dynamically as TIBCO iProcess Objects Server instances are started and stopped. The TIBCO iProcess Objects Director maintains this list by checking the **process_config** table of the iProcess Engine to which it is associated.

For information about using and configuring the TIBCO iProcess Objects Director, see the *TIBCO iProcess Objects Director Administrator's Guide*.

2

Naming Conventions

Naming Conventions Used in TIBCO iProcess Server Objects (Java)

The following naming conventions have been used in the TIBCO iProcess Server Objects (Java) product:

Object Names

- All non-XML Server Object names begin with "**s**" (e.g., sNodeManager).
- All XML Server Objects name begin with "**x**" (e.g., xNodeManager).
- All Value Object names begin with "**v**" (e.g., vCase). Another naming convention used with Value Objects indicate the amount of information returned by the object:
  - **v<*type*>Id** - Returns enough data to identify the object instance (e.g., vNodeId).
  - **v<*type*>** - Returns the data most commonly required by the user of the object (e.g., vNode).
  - **vA<*type*>** - Returns all data available for the object (e.g., vANode).
- Value Objects whose names end in "**Def**" reference "metadata", i.e., static data that contains the definition of the object (for instance, the definition of a procedure (vProcDef)).
- The names of Enumeration Type Objects begin with "**SW**" and end with "**Type**" — for example, SWAttributeType.
- Error Objects use the naming convention "**vEx<*Object*>**", where <*Object*> is the type of object that was either being passed as a parameter or returned by the method when the error occurred — for example, vExAttribute.
- Content Request Objects use the naming convention "**v<*type*>Content**", where <*type*> is the type of object whose content you are specifying — for example, vCaseContent.
- "List state" objects use the naming convention "**v<*type*>ListState**", where <*type*> is the type of object in the list. List state objects are returned by the single-block access list methods (**make**<*type*>**List** and **fetch**<*type*>**List** methods).

Method Names

- All method names on **Value Objects** begin with one of the following two names:
  - **get** - Returns the value noted in the method name (e.g., getWorkItemTag).
  - **is** - Returns a Boolean value used as a flag.
- Method names beginning with "**get**" and ending in "**s**" (plural) return an array of Java objects (e.g., getWorkItemFields).
- Method names ending in "**Cnt**" return a "count" (e.g., getDeadlineCnt).

- Method names ending in "**List**" return a pageable list (e.g., getWorkItemList). See *Working with Lists on page 137* for more information about pageable lists.
- Method names ending in "**ListHeld**" return a "held" pageable list (e.g., getWorkItemListHeld). See *Working with Lists on page 137* for more information about held pageable lists.

Parameter Names

- Parameter names beginning with "**a**" are input parameters (arguments) (e.g., aWorkQTag).
- Parameters that are followed by "**[ ]**" indicate that an array of objects is expected as input (aWorkQTags[ ]).

3

Object Types

Object Types

TIBCO iProcess Server Objects (Java) comprise the following types of objects:

- **Server Objects -** These objects implement the methods the client uses to request data (in Value Objects) and to initiate changes within the TIBCO iProcess Engine. Server Objects contain only methods — they do not contain any data.

  The names of Server Objects begin with "**s**" — for example, **sUser**.

- **Delegate Objects -** The RMI and EJB configurations use Delegate Objects — in these configurations, the client application instantiates a Delegate Object, which in turn causes a stub for the applicable Server Object to be sent to the client. The client makes method calls via the remote stub object. These objects make client code portable between all of the TIBCO iProcess Server Objects (Java) configurations (standalone, RMI, and EJB).

  The names of Delegate Objects begin with "**s**" — for example, **sUser**. (They look and act just like a Server Object.)

- **XML Server Objects -** These objects allow you to retrieve all data from the TIBCO iProcess Engine as an XML data stream, rather than in object format. For more information, see *XML Interface* on page 294.

  The names of the XML Server Objects begin with "**x**" — for example, **xUser**.

- **Value Objects -** These are data objects. They return data from the TIBCO iProcess Engine to the application when a request is made for the data through a method call on a Server Object. Some Value Objects can also be constructed for use as an input parameter to a Server Object method (see *Constructing Value Objects* on page 15 for more information).

  The names of Value Objects begin with "**v**" — for example, **vWorkItem**.

- **Content Request Objects -** These objects are used to specify how much "content", in the form of dependent objects, to return with requested Value Objects. For example, the **vGroupContent** object contains the **isWithAttributes** flag. This flag specifies whether or not **vAttribute** objects are returned with the requested **vGroup** objects. This allows you to prevent data from being retrieved from the server if you don't need it. See *Retrieving Dependent Objects on page 160* for more information about using Content Request Objects.

  Content Request Objects use the naming convention "**v<Object>Content**", where <Object> is the name of the object whose content you are specifying — for example, **vCaseContent**.

- **Error Objects -** These objects are used to handle errors that may occur as a result of a method call that either passes multiple objects as parameters, or returns multiple objects. The Error Objects provide a means for the method to continue processing when an error occurs with one or more of the items, without having to abort the entire operation. See *Error Handling* on page 281 for more information.

Error Objects use the naming convention "**vEx<Object>**", where <Object> is the name of the object that is either being passed as a parameter or returned by the method when the error occurred — for example, **vExAttribute**.

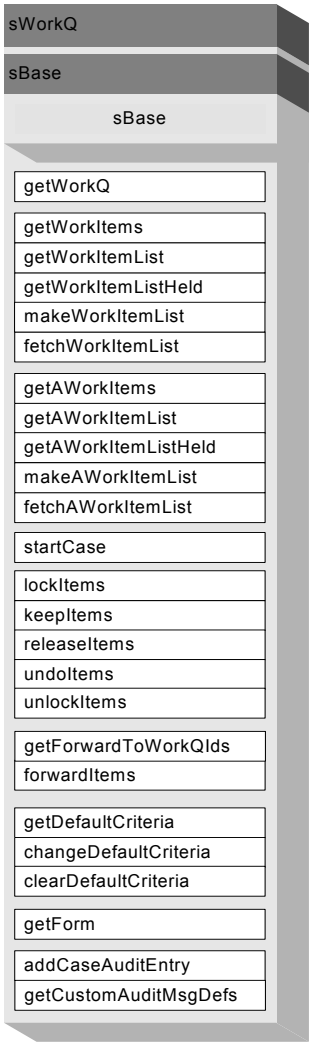
- **Criteria Objects** - There are two criteria objects: **vACaseCriteria** and **vWICriteria**. These objects are used to specify filter and sort criteria when requesting work items or cases from the server. They allow you to limit the number of items returned, as well as specify the order in which they are returned. See the appropriate *Filtering Work Items and Cases* chapter on page 193, page 221, or page 247 and *Sorting Work Items and Cases* **on page 273** for more information.
- **Enumeration Type Objects** - These objects provide the ability for "readable words" to be used in code, rather than numbers or characters. For example, the **SWAuditActionType** Enumeration Type Object equates swStartCase to 0, swProcessedTo to 1, and so on. In code, "swStartCase" can be used instead of "0", which provides for more readable code. Enumerations are both passed as parameters to method calls and returned by method calls.

  The names of Enumeration Type Objects begin with "**SW**" and end with "**Type**" — for example, **SWAttributeType**.

The following subsections provide more detail about the Server and Value Objects.

Server Objects

Server Objects provide a thin Java wrapper around C++ code that formats the request from the client to the TIBCO iProcess Objects Server. These objects basically act as a pass-through for data from the TIBCO iProcess Objects Server to the client, and vice versa. The name of all Server Objects begin with a lower-case "**s**".



In a stand-alone configuration, the client application instantiates the desired Server Object so that methods can be called to obtain one or more Value Objects.

In RMI and EJB configurations, there is a corresponding **Delegate Object** for each of the Server Objects that can be instantiated. The client application instantiates the appropriate Delegate Object, then calls the desired methods. It appears the same as from the stand-alone configuration since the names of the Server Objects and their corresponding Delegate Objects are the same.

Server objects contain only methods — they do not hold any data (although they do hold "session" data, maintaining a user session with the server). Client applications make calls to the methods on Server Objects. The Server Object passes on the request to the TIBCO iProcess Objects Server, which may return data in the form of a Value Object.

The Server Objects have been divided into logical areas of functionality. Most of them can be constructed publicly — some, however, are constructed only by other objects (e.g., sPageableList). The following is a list of the Server Objects that have public constructors:

- **sNodeManager** - This is used for server discovery — it uses the UDP broadcast mechanism to locate and identify available TIBCO iProcess Objects Servers.
- **sNode** - This is used to configure elements of a node, such as users, groups, attributes, roles, etc.
- **sProcManager** - This provides access to procedure definitions, including definitions of the elements of procedures, such as steps, forms, and fields.
- **sUser** - This provides access to information that is relevant to a particular user. For example, you can access a user's work queue, determine which procedures the user has authority to start and audit, change the user's password, etc.
- **sCaseManager** - This allows you to list currently active cases, as well as perform functions on those cases, such as closing and purging.
- **sWorkQ** - This provides methods used to perform work queue functions, such as locking, keeping, and releasing work items in a work queue.
- **sWorkQManager** - This provides access to work queues on a node, to configure access to a work queue, and to set up work queue redirection and participation.
- **sSession** - This is used to create other Server Objects using the same user session as the Server Object from which the "create" method was called. This allows you to share the TCP connection among multiple Server Objects, rather than have a TCP connection for every Server Object that is created (see *Sharing User Sessions* on page 12).
- **sIPEConfig** - This object contains methods that allow you to either get the engine's database configuration information, or to get/set the activity monitoring configuration.

The Server Objects have been designed to provide you with the information needed (in the form of Value Objects) to perform a particular task or function.

Note that each of the Server Objects has an equivalent "XML" Server Object that returns data in an XML data stream rather than in an object format. For more information, see *XML Interface* on page 294.

Garbage Collection for Server Objects

When using RMI-IIOP, remote objects that are created in the Server Factory (see *Server Factory* on page 20 for more information) must be explicitly removed for them to become available for garbage collection. A **remove** method is provided on each of the Server Objects listed in the previous section (plus on **sPageableList**) for the purpose of flagging the object for removal. The **remove** method should be called when a remote object is no longer needed. This releases the memory resources used by the object.

An object timeout mechanism (called the **objectMonitor**) is included in the **rsServerFactoryImpl** class to ensure that remote objects that are not explicitly removed by calling the **remove** method will eventually timeout, reclaiming system memory. The **objectMonitor** should only be used when running RMI using the IIOP protocol. It should not be started when running under EJB because the EJB container has its own timeout mechanism. (See *Command-Line Options* on page 20 for more information about the **objectMonitor**.)

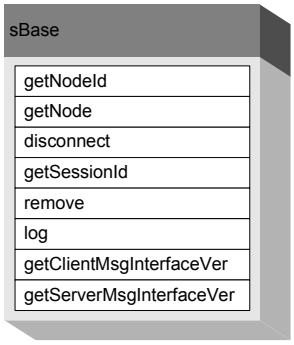
What is a User Session?

All of the Server Objects (with the exception of **sNodeManager**, **sClientLog**, and **sPageableList**) cause a user session to be started when they are instantiated. The user session represents a connection between a Server Object and a TIBCO iProcess Objects Server. For every user session, there is a TCP connection along with a user login to the TIBCO iProcess Objects Server.

The public constructor for each of the Server Objects that starts a user session requires a **vNodeId** Value Object, a user name, and a password. For example:

```
sWorkQManager(vNodeId aNodeId,
                String aUserName,
                String aPassword)
```

This provides the TIBCO iProcess Objects Server with the information it needs to establish a TCP connection and user login to a specific node (TIBCO iProcess Objects Server).



The **sBase** object, which is inherited by those Server Objects that start user sessions, provides a **disconnect** method that should be called to close the TCP connection and optionally log the user out of the node (i.e., free server resources) when that session is no longer needed — see *Disconnecting User Sessions* on page 12. Sessions are expected to be short lived. Server Objects should be used in the context of a single web page and re-created on each web page as needed to access the TIBCO iProcess Objects Server.

The **getSessionId** method returns the user session ID, for information purposes only. This ID, which is the authentication token for the Server Object, cannot be passed to any other Server Object to bypass the authentication process performed at object creation. The **sClientLog** object does not use a user session (i.e.,

no TCP or login). The **sNodeManager** object creates a user session when in methods **setSrvLogOptions**, **resetSrvLog** and **getANode**. The **sPageableList** object shares the user session from the Server Object where the method used to create the pageable list is called.

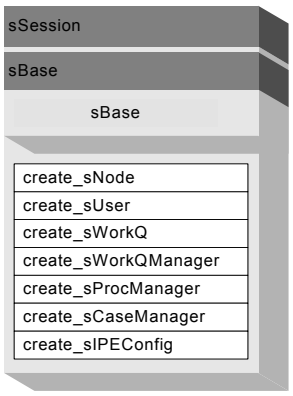
Disconnecting User Sessions

When a user session is no longer needed, you should call the **disconnect** method on **sBase** to close the TCP connection, as follows:

- call **disconnect**() or **disconnect**(False) (i.e., with no parameter or with the *releaseAllResources* parameter set to False). This closes the TCP connection (socket) between the Server Object and the TIBCO iProcess Objects Server. It is strongly recommended that **disconnect**(False) be called explicitly since TCP connections will remain open until the finalization is run in the gc thread. Since it may take some time for an object to be garbage collected/finalized, idle TCP connections are unavailable for re-use. This could end up a large number depending on the application and the efficiency of the garbage collection. The user's context on the TIBCO iProcess Objects Server is preserved and efficiently re-attached on subsequent client connections (creation of Server Objects).
- call **disconnect**(True) (i.e., with the *releaseAllResources* parameter set to True). This releases both client-side and server-side resources. This should only be done when a user is no longer using an application. This should NOT be called as part of releasing Server Objects on each web page, as performance would be adversely affected due to the overhead of creating a new user login session on the TIBCO iProcess Object Server. Held pageable lists associated with the user session will be released.

Sharing User Sessions

The **sSession** Server Object is provided to allow you to share user sessions within the context of one end-user. For example, if multiple Server Objects are needed in the context of a web page, the **sSession** object would be used for creating the Server Objects used on the web page. This minimizes the number of login requests and TCP connections used within a web page. If an **sSession** object is persisted at the Application level, it should always be associated with a particular end-user. **sSession** objects should NOT be pooled or used to share logins/TCP connections among multiple end-users. Sharing **sSession** objects between different end-users will adversely affect performance since all work is being done on behalf of a single user login to the TIBCO iProcss Object Server.



The **sSession** object contains “create” methods that are used to create multiple Server Objects that all share the same TCP connection and login to the TIBCO iProcess Obects Server node. For example, calling the **create_sUser** method causes an **sUser** object to be created that will share the user session with the existing **sSession** object. Any additional Server Objects that are created from that **sSession** object will also share the same user session/connection.

Note that if multiple Server Objects share a session, calling **disconnect** closes the TCP connection to the TIBCO iProcess Objects Server for ALL Server Objects that are sharing the session. If **disconnect** is not explicitly called, the TCP connection will be closed when the last Server Object using it is destructed.

There is one other place in TIBCO iProcess Server Objects (Java) where you can share a user session: the **sUser** object contains a **create_sWorkQ** method that does the same thing as the methods on **sSession** — it creates an **sWorkQ** object that shares a user session with the **sUser** object from which it was called.

Server Object Parameters

Many of the method calls from Server Objects require that parameters be passed with the call. The following conventions are used in the TIBCO iProcess Server Objects (Java) documentation to describe these parameters.

- By convention, method parameters begin with an "**a**" to identify them as *arguments* that require an input value. The example below shows a method that requires two arguments, *aWorkQTag* (which requires a String) and *aRedirection* (which requires a **vRedirection** object).

```
void changeRedirection(String aWorkQTag,
                       vRedirection aRedirection)
```

- A "[ ]" in an input or return parameter specifies that an array of items is required as input or returned, respectively. Note that methods that expect an array of Value Objects as input are not prototyped to accept a single instance. To pass a single object, it must be placed on an array, then the array containing the single object can be passed as a parameter. The example below shows a method that expects an array of Strings as an input parameter, and returns an array of **vWorkItem** objects.

```
vWorkItem[] getWorkItems(String[] aWorkItemTags,
                         vWIContent aWIContent)
```

Delegate Objects

Each of the Server Objects that can be instantiated (i.e., they have a public constructor), has a corresponding Delegate Object. Client applications running in an RMI or EJB configuration actually instantiate the Delegate Object instead of the desired Server Object. (Stand-alone configurations don't use Delegate Objects.)

Delegate Objects are available for the following Server Objects:

- sCaseManager
- sNode
- sNodeManager
- sProcManager
- sSession
- sUser
- sWorkQ
- sWorkQManager
- sIPEConfig

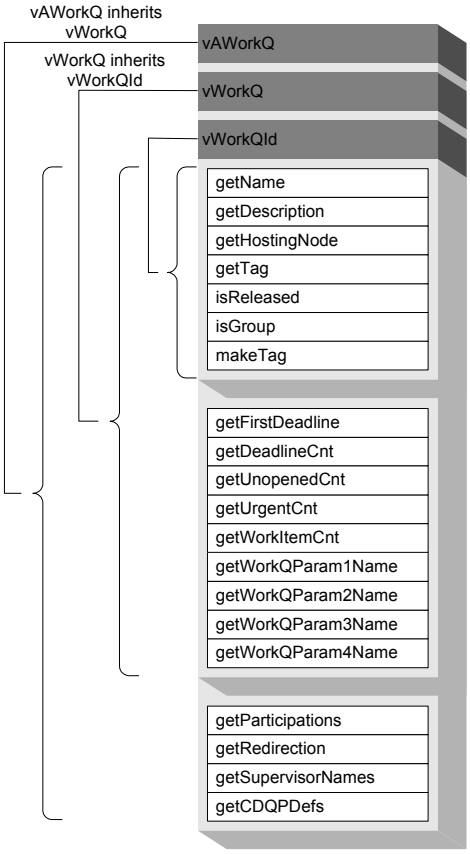
See the on-line help for information about the constructors for these objects.

After instantiating a Delegate Object, the server returns a remote stub object to the Delegate Object. The client's method calls against the Delegate are passed through the remote stub. From the client's perspective, however, it is making method calls against the methods on the Server Object.

Value Objects

Value Objects provide a snap-shot of data that was returned from the TIBCO iProcess Objects Server. There is no means of "refreshing" the Value Object's data. To get a new snap-shot of the data, the client must request another Value Object by making another method call. The name of all Value Objects begin with a lower-case "**v**".

Value Objects implement inheritance — the object model graphics illustrate which subclasses inherit their superclasses, as shown in the illustration on the right.



The client can limit the amount of data returned in the Value Objects to only the amount of data needed to perform the required task. **Requesting unneeded data affects the performance of the application.** The naming conventions used for Value Objects identify the type and amount of data that is returned in the object:

- **v<Object>Id** - Returns enough data to identify the object instance.
- **v<Object>** - Returns the data that is most commonly requested by the user.
- **vA<Object>** - Returns all data available for the object.

Note that using the "content" parameters when requesting data also allows you to limit the amount of data that is retrieved from the server. See *Retrieving Dependent Objects on page 160* for more information.

Serialization

All Value Objects implement the **java.io.Serializable** interface. This allows the RMI and EJB (which uses RMI) solutions to marshall Value Objects across the RMI boundary.

Multiple Value Objects

If a method call results in multiple Value Objects being returned to the client, they will be returned in one of two ways:

- **An Array** - If the method name begins with "get" and ends in an "s", it will return either an array of String objects or an array of Value Objects. Examples are shown below:

```
String[] getSupervisorNames(String aWorkQTag)

vParticipation[] getParticipations(String aWorkQTag)
```

  Arrays are used when the number of Value Objects expected to be returned is typically not a large number.

- **A Pageable List** - If the method name begins with “get” and ends with “List”, it returns a “pageable list”, which is a special Server Object (**sPageableListR**) that is used to handle a potentially large numbers of objects.

  ```
  sPageableListR getWorkItemList()
  ```

  The pageable list is used when the number of Value Objects expected to be returned can be very large. It contains methods that allow you to retrieve a specified number of Value Objects, rather than all available objects. (See *Working with Lists on page 137* for more information.)

Constructing Value Objects

Many of the methods on Server Objects require a Value Object or an array of Value Objects to be passed as input parameters. For this reason, many of the Value Objects have public constructors. For example, the **changeAttributes** method (shown below) requires an array of **vAttribute** objects as an input parameter:

```
void ChangeAttributes(string[] aUserNames,
                      vAttribute[] aChangedAttributes)
```

Therefore, the **vAttribute** object has a public constructor that allows you to create the object:

```
public vAttribute(string aName,
                  Object aValue,
                  SWAttributeType aType)
```

The on-line help system provides constructor information for those Value Objects whose constructors are public.

Some Value Objects do not have public constructors. You are expected to obtain these Value Objects by making a method call on a Server Object that returns the desired Value Object. The on-line help system also contains cross-reference information about which methods return each of the Value Objects.

Are Objects Reentrant?

Since TIBCO iProcess Server Objects (Java) was designed to support web-based applications, there is no state held except the connection information in the Server Objects. Therefore, all Server Objects (except the pageable list objects (sPageableList, sPageableListR, and sPageableListJ) and the sSession object) and all Value Objects are reentrant. Since each method on the Server Objects creates its own message to the TIBCO iProcess Objects Server, sharing them between threads should not be a problem.

4

Interfaces

Introduction

TIBCO iProcess Server Objects (Java) has one true Java interface — the **SI Interface** (for "Server Interface"). There are two implementations of the SI Interface available:

- **JBase** - This is the local solution — it exposes TIBCO iProcess Server Objects functionality as simple Java objects. This interface does not make use of remote objects. It is typically used when incorporating a broker application that is automatically processing work items arriving in a particular work queue.
- **RMI** - This is the remote solution — it uses Java's Remote Method Invocation (RMI) technology. This allows the client to be located on a machine remote from the TIBCO iProcess Server Objects.

  Note - There is also an Enterprise JavaBeans (EJB) implementation of TIBCO iProcess Server Objects available, which provides the ability to perform business logic functions through EJBs in the middle tier. This architecture, which makes use of a Java application server for scalability, conforms to the J2EE specification. Note, however, that as of version 10.3.0 of the TIBCO iProcess Server Objects (Java), the EJB implementation is no longer being updated with new features and enhancements. It is still available, but is being phased out of the product line.

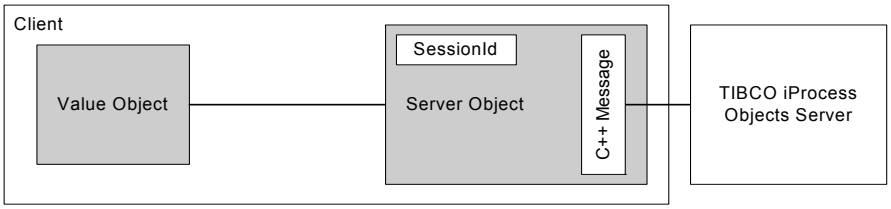
The interfaces described above provide data in the form of objects. There is also a XML interface available that allows you to retrieve all data from the TIBCO iProcess Engine as an XML data stream rather than in the form of objects. For more information about the XML interface, see *XML Interface* on page 294.

JBase Implementation

The JBase implementation exposes functionality as simple Java interfaces — it does not make use of remote objects. This implementation would typically be used with a multi-threaded application running on the server, processing work items arriving in a particular work queue. It provides the fastest and most efficient use of TIBCO iProcess Server Objects (Java) because there are no remote or out-of-process links (jumps).

Process Flow

The following illustrates the use of JBase.



The client application directly calls methods on Server Objects, which in turn, call C++ code to request data from the TIBCO iProcess Objects Server. The TIBCO iProcess Objects Server returns the requested data to the Server Object, which returns the data in the form of a Value Object to the client.

Packages

Jbase is contained in the following packages:

- **com.staffware.sso.data** - Contains the Value Objects.
- **com.staffware.sso.jbase** - Contains the Server Objects.
- **com.staffware.sso.util** - Contains utility functions.

JAR File

All classes for JBase are provided in the **ssoJBase.jar** file.

RMI Implementation

The Remote Method Invocation (RMI) implementation provides the ability for a server to create remote objects, make them accessible across a network, then wait for a client application to invoke methods on the remote objects. RMI handles all of the details of communication between the virtual machines on the client and server. To the client programmer, invoking methods on a remote object looks like a standard Java method invocation.

RMI is supported across either the standard Java Remote Method Protocol (JRMP) or the CORBA standard Internet Inter-Orb Protocol (IIOP). The RMI Delegates use the IIOP protocol by default, but can be instantiated to use the JRMP protocol.

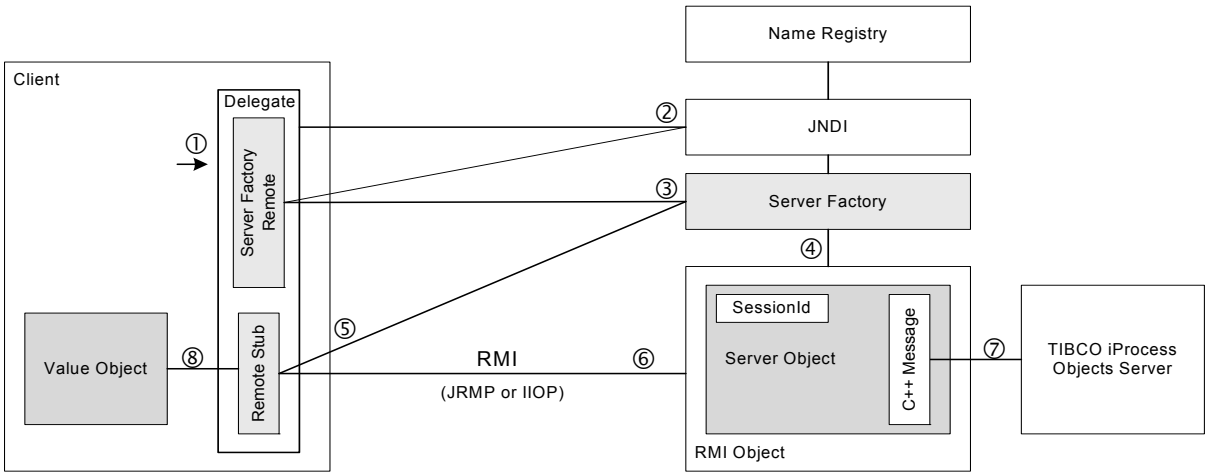
RMI does not register its server classes directly. This is to prevent all clients from accessing the same server objects, which could result in a potential processing bottleneck. Instead, an **rsServerFactory** class is registered. The default binding name for the Server Factory is "**ssoServerFactory**". This factory is used by all clients to create Server Object instances. In this way, each client has its own server class.

The system can be configured to have multiple Server Factories running and registered under different names. The system administrator must ensure that the Server Factory is installed and running before clients try to access data through the RMI and EJB interfaces. The Server Factory is not self-initiating.

TIBCO iProcess Server Objects (Java) uses the Java Naming and Directory Interface (JNDI) interfaces to access the remote naming services. The registration of the RMI Server must match the access method of the RMI client (see *Delegate Objects* on page 23).

Process Flow

The following illustrates the process flow of RMI.



1. The **client application** instantiates the **Delegate Object** that corresponds to the Server Object from which the application wants to invoke methods.

   When constructing the Delegate Object, the client must pass node, user, and password information. This is used to specify which node (TIBCO iProcess Objects Server) to which the Server Object is to connect, as well as the information needed to authenticate the user on the node.

2. The **Delegate Object** does a lookup in the **Name Registry** through **JNDI** for a **Server Factory** object, which is returned to the delegate. Note that this lookup for the Server Factory object is done only the first time you instantiate a Delegate Object. The Server Factory object will then be maintained as a static variable. All subsequent requests will simply get the object handle back.
3. The **Delegate Object** asks the **Server Factory** to create the desired remote Server Object.
4. The **Server Factory** instantiates the **Server Object** on the remote machine.
5. The **Server Factory** returns a **remote stub** to the client. This is the object from which method calls are actually made, although it appears to the client as though they are made on the originally instantiated Server/Delegate Object.
6. As method calls are made via the **remote stub**, the method parameters are marshalled across the RMI boundary.
7. The **Server Object** passes the request to the TIBCO iProcess Objects Server, which returns the requested data to the Server Object.
8. The **Server Object** sends the data to the client in the form of a **Value Object**.

In Short: The client instantiates a **Delegate Object**, makes method calls against it, and receives a **Value Object** in return.

Packages

RMI is contained in the following packages:

- **com.staffware.sso.data** - Contains the Value Objects.
- **com.staffware.sso.jbase** - Contains the Server Objects.
- **com.staffware.sso.util** - Contains utility functions.
- **com.staffware.sso.rmi** - Contains the Delegate Objects, stubs, ties, and RMI implementations.

JAR File

All classes required for RMI access are included in the **ssoRMI.jar** file. The **ssoJBase.jar** file is not required.

The same JAR file is used for both the client and server. This ensures that the RMI interfaces match the base objects, and that the client's interfaces match the server implementations.

RMI Components

RMI is composed of the following types of files:

On the server:

- **rs<***ObjectName***>.class** - Remote Interface
- **rs<***ObjectName***>Impl.class** - Remote Interface Implementation
- **rs<***ObjectName***>_tie.class** - Server-side ties (IIOP only)
- **rsServerFactory.class** - Server Factory
- **rsServerFactoryImpl.class** - Server Factory Implementation

On the client:

- **s<***ObjectName***>.class** - Delegate Object
- **s<***ObjectName***>_stub.class** - Proxy for remote object

These file types are described below.

Remote Interface

The remote interface defines the methods that a client will be able to remotely invoke from the Server Object.

The naming convention for remote interface files is **rs<***ObjectName***>.class** (e.g., **rsProcManager.class**).

Remote Interface Implementation

Each of the remote interfaces has a corresponding implementation class. The implementation class provides an implementation for each of the remote methods in the remote interface, and defines the constructor for the remote object.

The naming convention for the implementation class files is **rs<***ObjectName***>Impl.class** (e.g., **rsProcManagerImpl.class**).

Each implementation class "exports" the object with the RMI implementation, which creates an identifier used by the stub to forward calls to the Server Object. This makes the object remotely accessible.

Whether the object exports the JRMP or the IIOP protocol is determined by how the Server Factory is instantiated. The default is to export IIOP for all objects created. Using the "**-protocol=JRMP**" command-line option when the Server Factory is started causes it to create objects using JRMP (see the "Server Factory" section below for more information).

If a server returns remote interfaces (i.e., the methods on **sSession** and **sUser.create_sWorkQ**), they will be returned using the same protocol with which the server was instantiated.

Server Factory

The **Server Factory** is a remote interface that contains methods to create the Server Objects. This remote interface has the name, **rsServerFactory.class**. The Server Factory must be running before the RMI objects are accessed. When the Server Factory is started, it will register itself with a JNDI naming service. The naming service and bind name are configurable.

The **Server Factory implementation** provides an implementation of each of the "create Server Object" methods in the Server Factory. This implementation class has the name, **rsServerFactoryImpl.class**. The Server Factory implementation passes a remote stub for the applicable Delegate Object that was instantiated on the client, which can then make method calls against that Delegate Object.

The **rsServerFactory** object contains a main() method and should be run as its own process:

```
java -Djava.security.policy="c:\ssoRMI\policy"
com.staffware.sso.rmi.rsServerFactoryImpl
```

Command-Line Options

There are a number of command-line options that may be included when the Server Factory is started. These include: **bind name, external rebind**, **export protocol**, **verbose output**, and **object timeout monitor**. These are described below:

- **Bind Name -** By default, the Server Factory is registered under the name, **ssoServerFactory**. However, this name can be overridden by including the "**-name**" command-line option when starting the Server Factory. This must be in the form:

  ```
  -name=<bind name in Registry>
  ```

- **External Rebind -** This command-line argument causes the class to not rebind to the naming service. When using WebSphere JAAS authentication, you must disable the internal rebind and allow an external rebind to the **rsServerFactoryImpl**. This is because with WebSphere security enabled for binding to the naming service, the **InitialContext rebind()** method cannot be called outside of a "privileged" context. For more information, see *External Rebind Needed When Using Either WebSphere JAAS or JBoss* on page 24.

- **Export Protocol -** This specifies the protocol used to marshall objects. The allowable protocols are IIOP or JRMP. By default, IIOP is used. To use JRMP, use this command-line option when starting the Server Factory. Note that the protocol that is used to start the Server Factory MUST be the same protocol used when constructing the Delegate Objects (see *Delegate Objects* on page 23). This is specified on the command line by including the "**-protocol**" option. This must be in the form:

  ```
  -protocol=<IIOP or JRMP>
  ```

- **Verbose Output**- This option causes information concerning the Server Factory to be sent to standard output. This is specified by including the "**-verbose**" option on the command line. This must be in the form:

  ```
  -verbose[:jndi|object|monitor]
  ```

  where:

  ```
  -verbose:jndi
  ```

  provides information about the JNDI environment. Example output:

  ```
  JNDI Context Environment:

  {java.naming.provider.url=<protocol://host#:port#>, java.naming.fac-
  tory.initial=<InitialContextFactory class>}
  ```

  ```
  -verbose:object
  ```

  provides log information for each object that is created by the Server Factory. Example output:

  ```
  Create an rsUser. (Node:StaffwareTest, User:user0001)
  ```

  ```
  -verbose:monitor
  ```

  provides details about the operation of the ObjectMonitor. Example output:

  ```
  yyyyMMddhhmmss ObjectMonitor thread: ---------------------------------------
  Thread pass: 5, 7ms., removed: 0, timedOut: 18, notTimedOut: 294
  total removed: 0, total TimedOut: 18
  Min Ave Max age timedOut: 604584 624281 646033
  Ave age removed: 0
  Ave age not timedOut: 226328
  ```

  If the **-verbose** option is used without parameters, all available information is written to standard output.

- **Object Timeout Monitor** - The object timeout monitor is used to cause remote objects to be automatically removed if they are not accessed in a specified period of time, reclaiming system memory. (Note - The object timeout monitor should only be used when running RMI using the IIOP protocol. It should not be started when running under EJB because the EJB container has its own timeout mechanism.)

  The object timeout monitor option consists of two arguments:

  ```
  -objectTimeout=<timeout time in seconds>
  -objectMonitor
  ```

  The **-objectTimeout** argument is used to specify the number of seconds in which an object should be removed if has not been accessed in that period of time. The default is 600 seconds (10 minutes). The **-objectMonitor** argument causes the object timeout monitor to be started. The object monitor is not started by default.

The following is an example using the available command-line options:

```
java -Djava.security.policy="c:\ssoRMI\policy"
com.staffware.sso.rmi.rsServerFactoryImpl
```

```
-name=ssoServerFactory1
-protocol=IIOP
-verbose:monitor
-objectTimeout=1200
-objectMonitor
```

Note that starting the **rsServerFactoryImpl** class with **-?** or **-help** displays a list of all available command-line options.

Registry Location

The location of the Registry is defined in JNDI by the Provider URL (**java.naming.provider.url**). There are several ways to indicate to the JNDI interface what the Provider URL should be. One is to include a **JNDI.Properties** file in the **classpath** and include the setting. Another is to pass it as a parameter to the JVM when it is started.

```
java -Djava.security.policy="c:\ssoRMI\policy"
-Djava.naming.provider.url=<Registry Provider URL>
com.staffware.sso.rmi.rsServerFactoryImpl
```

The following are examples for <Registry Provider URL>:

- RMIRegistry running on your local machine on port 1099:

  ```
  rmi://localhost:1099
  ```

- COS registry running on an external machine port 900, J2EE Reference Implementation naming server on external machine:

  ```
  iiop://10.20.30.103:900
  ```

Naming Factory

This tells the registry what type of objects to create when this object is requested. This is a JNDI context object factory; the class specified determines what type of registry is being accessed by the Provider URL. The class factory is defined in JNDI by the initial factory (**java.naming.factory.initial**). For example:

```
java -Djava.security.policy="c:\ssoRMI\policy"
-Djava.naming.provider.url=<Registry Provider URL>
-Djava.naming.factory.initial=<Registry Context Factory>
com.staffware.sso.rmi.rsServerFactoryImpl
```

The following are examples for <Registry Context Factory>:

- For accessing RMIRegistry:

  ```
  com.sun.jndi.rmi.registry.RegistryContextFactory
  ```

- For accessing COS naming server:

  ```
  com.sun.jndi.cosnaming.CNCtxFactory
  ```

JNDI must be configured for the Server Factory to properly register. Matching values should be used by the client (Delegate) to locate the registered Server Factory.

Note - For details about JNDI, see: http://java.sun.com/products/jndi/tutorial/.

Delegate Objects

There is a "Delegate Object" for each of the Server Objects. It is a Delegate Object that you actually instantiate in your client application when you want to make a method call on a Server Object.

A Delegate Object acts the same as a stand-alone Server Object, and has the same naming convention: **s**<*ObjectName*>**.class**.

During the instantiation of the Delegate Object, it accesses a "Server Factory" object, which in turn creates the requested remote Server Object.

The export protocol (JRMP or IIOP) is determined by the *aSF_IsJRMP* parameter in the Delegate constructor. The default is to use IIOP. Note that the export protocol specified when the Delegate is constructed MUST be the same protocol specified when the Server Factory is started (see *Command-Line Options* on page 20).

The Delegates use the default JNDI Context to determine the registry URL and the initial factory class. To set these values, use the **JNDI.Properties** file, or pass **-D** values to the Java Virtual Machine when the client is started (see *Registry Location* on page 22). Remember that the values used here must match those used when the **rsServerFactory** was started.

Two constructors are provided for each Delegate Object, allowing you to use the one that is appropriate for your situation. Examples are shown below (using the **sUser** Delegate Object).

Constructor 1: This uses the default JNDI context, ssoServerFactory, and IIOP:

```
sUser(vNodeId aNodeId,
       String aUserName,
       String aPassword)
```

Constructor 2: This uses the default JNDI context, a specified Server Factory name (*aSF_Name*), and either JRMP or IIOP (if *aSF_IsJRMP*=True, use JRMP; if *aSF_IsJRMP*=False, use IIOP). It also includes a hashtable that contains explicit JNDI environment settings to use when creating the context used to locate the ssoServerFactory.

```
sUser(vNodeId aNodeId,
       String aUserName,
       String aPassword,
       String aSF_Name,
       boolean aSF_IsJRMP,
       Hashtable aSF_JNDIEnv)
```

Passing an empty string for the Server Factory name (*aSF_Name*="") means to use the default **ssoServerFactory**. Passing a NULL for the JNDI environment means to use the default JNDI context settings.

Application Servers

The RMI interface is J2EE compliant. Therefore, it can be run on any Application Server that meets the J2EE specification. Testing was performed on WebLogic. Note, however, that we do not recommend running the JBase interface under an Application Server for the following reasons:

- The JBase interface is not J2EE compliant — the J2EE/EJB specifications prohibit compliant applications from running native code; JBase makes native calls using JNI.
- The security offered by J2EE compliance is compromised.
- If the application crashes, so will the Application Server.

- Scalability is limited, as follows:
  - When running RMI, multiple **rsServerFactory** processes can be started as load requires and these can run on any machine. This is not possible using JBase.
  - Since the **rsServerFactory** runs in its own JVM if you use RMI, it can be tuned specifically for TIBCO iProcess Server Objects (Java) without concern for the requirements of other processes running under the same JVM. This is not possible with JBase.
- The application is not portable, as follows:
  - An application running under an Application Server is portable if it does not depend on O/S-specific libraries. With RMI, such an application is portable and can be deployed on an Application Server running on Windows, AIX, Sun, or HP-UX, for example, with no modification required. The **rsServerFactory** process runs in its own JVM and it alone loads the O/S-specific libraries. In contrast, if JBase is running under an Application Server, the application is no longer portable because it would require loading the O/S-specific libraries to run.

Note - The last three bullet items above are also valid reasons JBase should not be run under a web server.

External Rebind Needed When Using Either WebSphere JAAS or JBoss

If you are using either WebSphere JAAS, or JBoss, you must disable the internal rebind and allow an external rebind to the **rsServerFactoryImpl**. This is because with WebSphere or JBoss security enabled for binding to the naming service, the **InitialContext rebind()** method cannot be called outside of a “privileged” context. The following is provided to allow this:

- When the following command-line argument is specified, the class does not rebind to the naming service.

  ```
  -externalRebind
  ```

- The following static method is provided to get a handle to the **rsServerFactoryImp** instance.

  ```
  public static rsServerFactoryImpl getExternalRebindHandle()
  ```

  This returns the **rsServerFactoryImpl** object so that it can be externally bound to the naming service within the **PrivilegedAction** context of the authenticated subject.

For more information, reference one of the following two subsections, depending on whether you are using WebSphere JAAS or JBoss.

External Rebind for WebSphere JAAS

You need to create a class that performs the required WebSphere JAAS authentication to login. This class must call the **rsServerFactoryImpl.main(String[] args)** method (with the **-externalRebind** option), then externally bind the **rsServerFactoryImpl** object to the **InitialContext** in the required **PrivilegedAction** context.

The steps shown below provide a summary of the requirements to login to WebSphere using JAAS and bind the **rsServerFactoryImpl** object.

Note - The examples shown in the steps below were taken from the following website that explains WebSphere Java Authentication and Authorization Service (JAAS):

http://publib.boulder.ibm.com/infocenter/wasinfo/topic/com.ibm.websphere.base.doc/info/aes/ae/csec_jaas.html

1. Configure the server for JAAS. For information about how to do this, see the link above.
2. Create the **InitialContext** required for the **SecurityServer**:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.websphere.naming.
        WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
Context initialContext = new InitialContext(env);
Object obj = initialContext.lookup("");
```

3. Create a **LoginContext**. This can use one of three **CallbackHandler** classes or a custom handler class:
   - **Non-prompt:**

     ```
     LoginContext lc = new LoginContext("WSLogin",
            new WSCallbackHandlerImpl("userName", "realm", "password"));
     ```

   - **GUI prompt:**

     ```
     LoginContext lc = new LoginContext("WSLogin",
            new WSGUICallbackHandlerImpl());
     ```

   - **Stdin prompt:**

     ```
     LoginContext lc = new LoginContext("WSLogin",
            new WSStdinCallbackHandlerImpl());
     ```

4. Do the login:

```
lc.login();
```

5. Get the authenticated Subject object from the **LoginContext**:

```
Subject s = lc.getSubject();
```

6. Call **rsServerFactoryImpl.main(String[] args)** with the **-externalRebind** option.
   - All the normal command-line arguments are passed in here except for the **-name=***FactoryName* argument. The name used for binding is specified in the **rebind(**..**)** method in **RegisteredName** (see step 7).

     Note - If a name other than "ssoServerFactory" is used, the delegate classes will need to use methods that specify this name also.

   - This will initialize the **rsServerFactoryImpl** without doing the **ctx.rebind()** and allows access to the object via the static method:

     ```
     public static rsServerFactoryImpl getExternalRebindHandle()
     ```

7. Create a **PrivilegedAction** object. This will do the **context.rebind(**..**)**:

```
PrivilegedAction action = PrivilegedAction() {
    public Object run() {
```

```
            initialContext.rebind(RegisteredName,
                rsServerFactoryImpl.getExternalRebindHandle());
        }
    }
```

8. Do the **PrivilegedAction** using the authenticated subject:

```
WSSubject.doAs(s, action);
```

External Rebind for JBoss

The following example source code is provided to help guide you through the process of an external rebind when using JBoss. Modify as needed for your specific implementation. Also, refer to the section on external rebind for WebSphere on page 24 for further descriptions.

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import com.staffware.sso.rmi.rsServerFactoryImpl;

public class ssoServerFactoryCustom {

  /**
  * @param args
  */
  public static void main(String[] args) {

    String[] params = new String[4];
    params[0] = "-verbose";
    params[1] = "-protocol=JRMP";
    params[2] = "-name=ssoServerFactory";
    params[3] = "-externalrebind";

      try {

      Hashtable env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY,
           "org.jnp.interfaces.NamingContextFactory");
      env.put(Context.PROVIDER_URL, "jnp://localhost:1099");

      Context initialContext = new InitialContext(env);
      Object obj = initialContext.lookup("");

      rsServerFactoryImpl.main(params);
      initialContext.rebind("ssoServerFactory",
          rsServerFactoryImpl.getExternalRebindHandle());

    } catch (Exception e) {
      e.printStackTrace();
      System.exit(-1);
    }
  }
}
```

EJB Implementation

Note - As of release 10.3, the EJB interface will no longer be updated to include new features / enhancements.

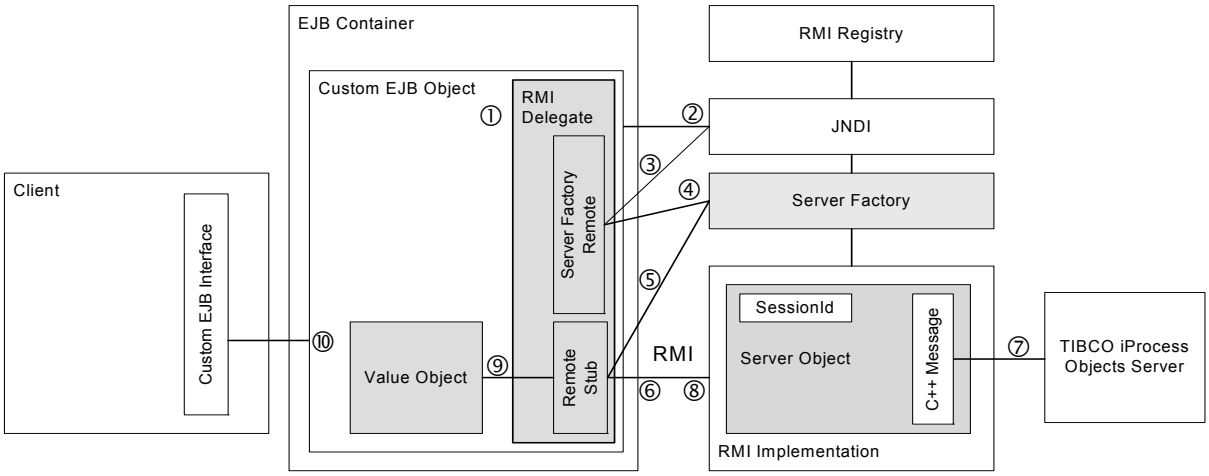
In an Enterprise JavaBeans (EJB) implementation, the RMI remote stubs are wrapped by EJB objects. This provides accessibility from the EJB container framework without having the JNI code of the underlying objects in play. The EJB objects are stateful session beans whose only piece of state is the RMI remote interface.

The following subsections illustrate and provide process flow for an EJB solution and a custom EJB solution.

Custom EJB Solution

Process Flow

The following illustrates the architecture of a custom EJB solution.



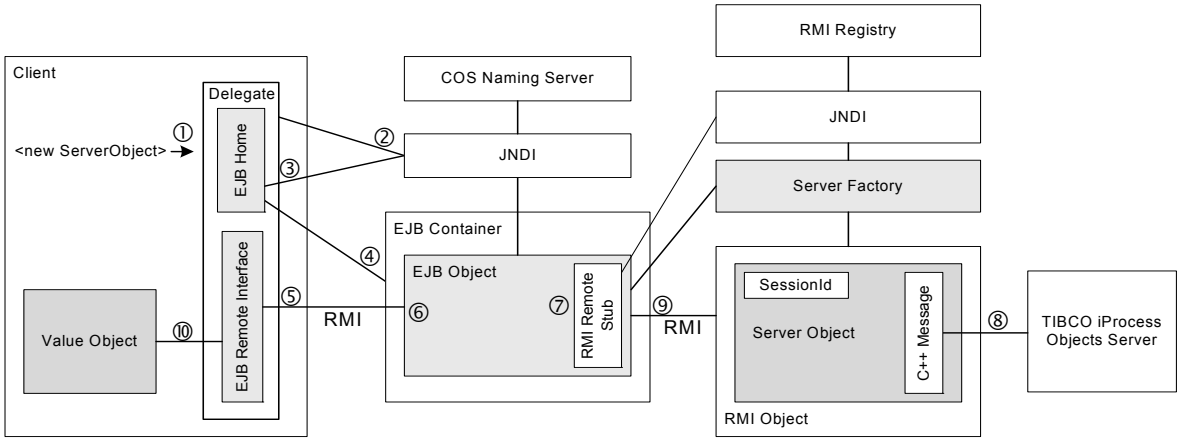
1. The **Custom EJB Object** instantiates an **RMI Delegate** that corresponds to the **Server Object** from which the application wants to invoke methods.
2. The **RMI Delegate** does a naming lookup via **JNDI** for the **Server Factory** object. (The Server Factory must be previously started.)
3. **JNDI** returns a remote reference to the **Server Factory**.
4. The **RMI Delegate** requests a **Server Object** from the **Server Factory**.
5. The **Server Factory** creates a **Server Object** and returns to the **RMI Delegate** a reference to the **Remote Stub**.
6. The **Custom EJB Object** makes method calls to the **RMI Delegate**, which in turn calls the **Remote Stub**. The **Remote Stub** then marshals the method parameters to the **Server Object** across the RMI interface.
7. The **Server Object** passes the request to the TIBCO iProcess Objects Server, which returns the requested data.

8. The **Server Object** returns the data in the form of a **Value Object** marshaled back across the RMI interface.
9. The **Remote Stub** returns the **Value Object** to the **Custom EJB Object**.
10. The **Custom EJB Object** then groups, filters, combines with other data or takes other appropriate actions to satisfy the requirements of its exposed functionality.

EJB Solution

Process Flow

The following illustrates the architecture of the EJB solution.



1. The **client application** instantiates the **Delegate Object** that corresponds to the Server Object from which the application wants to invoke methods.
2. The **Delegate Object** does a naming lookup in **JNDI** for the **EJBHome** object.
3. **JNDI** returns a reference to the **EJBHome** object.
4. The Delegate calls **home.create**, which creates an **EJB Object** in the **EJB container**. This instantiation of the EJB Object is considered the **Session Bean**.
5. A reference to the **EJB Object** in the container is given to the **EJB Remote Interface** on the client.
6. The client makes **method calls** to the **EJB Remote Interface**, which marshals the method parameters to the **EJB Object** across the RMI interface.
7. The **EJB Object** invokes the appropriate method on the **RMI Remote stub**, which marshals the method parameters to the Server Object across the RMI interface.
8. The **Server Object** passes the request to the TIBCO iProcess Objects Server, which returns the requested data to the Server Object.
9. The **Server Object** sends the data in the form of a **Value Object** to the **EJB Object**.
10. The **EJB Object** passes the **Value Object** to the **client application**.

In Short: The client instantiates a **Delegate Object**, makes method calls against it, and receives a **Value Object** in return.

Packages

EJB is contained in the following package:

- **com.staffware.sso.ejb** - Contains the Value Objects, Server Objects, Delegate Objects and stubs.

EJB Components

EJB is composed of the following types of files:

On the server:

- **es<***ServerClassName***>Home.class** - EJBHome remote interface
- **es<***ServerClassName***>.class** - EJBObject remote interface
- **es<***ServerClassName***>Bean.class** - Enterprise bean implementation

On the client:

- **s<***ObjectName***>.class** - Delegate Object

The EJBObject and EJBHome remote interfaces and the Enterprise bean implementation are bundled together in a JAR file, which is given to the EJB container for deployment. The EJB container creates the EJBObject and EJBHome implementations, and sets up the execution environment so that Delegate Objects can be instantiated in the client application.

When deploying the beans, it is required that they be registered in a naming service registry in order for the EJB Delegate objects to be able to find them. Sample EJB/JNDI names are shown below:

- ssoUser
- ssoWorkQ
- ssoNode
- ssoNodeManager
- ssoPageableBase
- ssoWorkQManager
- ssoProcManager
- ssoSession
- ssoCaseManager

Also note that a deployment descriptor file (**ejb-jar.xml**) is required to deploy EJBs. The specific requirements will depend on your implementation.

For each server delegate class there are three classes that need to be referenced in the **ejb-jar.xml** to define the enterprise bean deployment descriptor:

- com.staffware.sso.ejb.es(*ServerClassName*)Home
- com.staffware.sso.ejb.es(*ServerClassName*)
- com.staffware.sso.ejb.es(*ServerClassName*)Bean

Example: For sCaseManager:

- com.staffware.sso.ejb.esCaseManagerHome (the home interface)
- com.staffware.sso.ejb.esCaseManager (is the remote interface)
- com.staffware.sso.ejb.esCaseManagerBean (is the bean implementation)

EJB deployment descriptor files are typically included in a WAR file under the META-INF path. The following example shows this for a WebLogic deployment:

```
META-INF
 | ejb-jar.xml
 | weblogic-ejb-jar.xml
 |
 └lib
    ssoEJB.jar
```

Example deployment descriptor files for WebLogic are included on the TIBCO iProcess Server Objects (Java) distribution CD.

EJBHome Remote Interface

There is one EJBHome remote interface file for each Server Object. This remote interface contains a method that creates the EJBObject for the desired Server Object.

The naming convention for EJBHome remote interface files is **es<***ObjectName***>Home.class** (e.g., **esProcManagerHome.class**).

EJBObject Remote Interface

The EJBObject remote interface defines the methods that a client will be able to remotely invoke from the Server Object.

The naming convention for remote interface files is **es<***ObjectName***>.class** (e.g., **esProcManager.class**).

The methods in the EJBObject remote interface must declare **RemoteException** in their throws clause. The RemoteException is thrown by the RMI system if a communication or protocol error occurs during the remote method call. This is caught by the Delegate and converted into a **vException** object.

Enterprise Bean Implementation

This is equivalent to the remote object implementation, except it doesn't explicitly implement the remote interface. Instead, it implements **javax.ejb.SessionBean**, which lets the container interact with the component.

When a method is called on the EJBObject, the method is forwarded by the container to the enterprise bean implementation.

The naming convention for enterprise bean implementation files is **es<***ObjectName***>Bean.class** (e.g., **esProcManagerBean.class**).

Delegate Objects

There is a "Delegate Object" for each of the Server Objects. It is a Delegate Object that you actually instantiate in your client application when you want to make a method call on a Server Object.

A Delegate Object acts the same as a Server Object, and has the same naming convention: **s**<*ObjectName*>**.class**.

During the instantiation of the Delegate Object, it accesses a "Server Factory" object, which in turn creates the requested remote Server Object.

The export protocol (JRMP or IIOP) is determined by the *aSF_IsJRMP* parameter in the Delegate constructor. The default is to use IIOP. Note that the export protocol specified when the Delegate is constructed MUST be the same protocol specified when the Server Factory is started (see *Command-Line Options* on page 20).

Two constructors are provided for each Delegate Object, allowing you to use the one that is appropriate for your situation. Examples are shown below (using the **sUser** Delegate Object).

Constructor 1: This uses the default JNDI context, ssoServerFactory, and IIOP:

```
sUser(vNodeId aNodeId,
       String aUserName,
       String aPassword)
```

Constructor 2: This uses the default JNDI context and a specified Server Factory name (*aSF_Name*), and either JRMP or IIOP (if *aSF_IsJRMP*=True, use JRMP; if *aSF_IsJRMP*=False, use IIOP). It also includes a hashtable that contains explicit JNDI environment settings to use when creating the context used to locate the ssoServerFactory.

```
sUser(vNodeId aNodeId,
       String aUserName,
       String aPassword,
       String aSF_Name,
       boolean aSF_IsJRMP,
       Hashtable aSF_JNDIEnv)
```

Passing an empty string for the Server Factory name (*aSF_Name*=“”) means to use the default **ssoServerFactory**. Passing a NULL for the JNDI environment means to use the default JNDI context settings.

5

Node Management

Introduction

Instantiating one of the Server/Delegate Objects requires that you provide the information needed to connect to a specific node (TIBCO iProcess Objects Server). At a minimum, this includes providing a **vNodeId** object, a user name, and a password when constructing the desired Server Object. The following example shows the constructor for the **sUser** object.

```
sUser(vNodeId aNodeId,
      String aUserName,
      String aPassword)
```

*Note - If the TIBCO iProcess Objects Server is configured to not require a user password, an empty string can be passed for the aPassword parameter. For information about turning on/off password checking by the TIBCO iProcess Objects Server, see the **SEOPasswordRequired** configuration parameter in the TIBCO iProcess Objects Server Administrator's Guide.*

*Note - You can alternatively construct a **vNodeCtx** object (which was added to the object model when the XML interface was added). The vNodeCtx object contains the context information needed for a connection to the server (vNodeId, username, and password).*

The **vNodeId** object that is passed in the Delegate Object constructor contains the physical connection information needed — name of the TIBCO iProcess Objects Server, IP address, TCP port, etc. The user name and password must be of a user who is currently defined on the node (TIBCO iProcess Objects Server) to which you are connecting.

Obtaining the **vNodeId** object needed to instantiate a Server Object can be accomplished in the following ways:

- **Construct the vNodeId object** - This requires that you know the name of the TIBCO iProcess Objects Server, name of the computer on which the TIBCO iProcess Objects Server software is installed, IP address, and the TCP port.
- **Send a directed UDP message to the node** - This verifies that a specific node exists and is available for use.
- **Send a UDP broadcast** - This is done to determine which nodes are on the LAN segment and available for use.
- **Use a TIBCO iProcess Objects Director** - You may also choose to use the TIBCO iProcess Objects Director to decide which TIBCO iProcess Objects Server to connect to. The TIBCO iProcess Objects Director is a standalone program that maintains a list of TIBCO iProcess Objects Servers that are currently available in a node cluster. When a client application (actually, a Server Object) needs access to a TIBCO iProcess Objects Server, it first establishes a connection to the TIBCO iProcess Objects Director. The TIBCO iProcess Objects Director then decides, based on a "pick method," which TIBCO iProcess Objects Server the client should

connect to. For more information about using TIBCO iProcess Objects Directors, see the *TIBCO iProcess Objects Director Administrator's Guide*.

Also note that the descriptions in the following sections that discuss creating vNodeId objects and sending UDP messages/broadcasts apply to both TIBCO iProcess Objects Servers and TIBCO iProcess Objects Directors; clients obtain a vNodeId object for a TIBCO iProcess Objects Director in the same way as for a TIBCO iProcess Objects Server.

These are described in the following subsections.

Constructing a vNodeId Object

The **vNodeId** object can be manually constructed if the following information is known:

- The computer name on which the TIBCO iProcess Objects Server or TIBCO iProcess Objects Director resides.
- The node name of the TIBCO iProcess Objects Server or TIBCO iProcess Objects Director.
- The IP address of the TIBCO iProcess Objects Server or TIBCO iProcess Objects Director. Note that this can be the host name, as long as the name resolves to the IP address of the machine on which the iProcess Objects Server or Director is running.
- The TCP port being used by the TIBCO iProcess Objects Server or TIBCO iProcess Objects Director.

The following is an example of constructing the **vNodeId** object, then using that Value Object to construct an **sUser** object.

```
vNodeId oNodeId;
String userName=”swadmin”;
String password=””;

String nodeName=”Helga”;
String computerName=”Olga”;
String ipAddr=”10.20.30.118”;
int tcpPort=12345;
boolean isDirector=false;
.
.
oNodeId = new vNodeId(nodeName, computerName, ipAddr, tcpPort, isDirector);
.
.
sUser oUser = new sUser(oNodeId, userName, password);
.
.
```

You can now call methods on the **sUser** object to retrieve the desired Value Objects.

Note - The **ipAddr** attribute in the **vNodeId** constructor supports the use of IPv6 addressing if the TIBCO iProcess Objects Server or Director uses an IPv6 address (an example IPv6 address is: he80::c531:b7a1:8922:ab12%13).

Sending a Directed UDP Message

You can send a directed User Datagram Protocol (UDP) message to a known node (TIBCO iProcess Objects Server or TIBCO iProcess Objects Director) to verify that the node is available. To send a directed UDP message, you must know the following:

- The node name of the TIBCO iProcess Objects Server or TIBCO iProcess Objects Director.
- Either the computer name on which the TIBCO iProcess Objects Server / Director resides, or its IP address. If the computer name is given, the client machine must be configured for TCP name resolution, typically either DNS or local host file.

To send a directed UDP message, construct an **sNodeManager** object, then call the **verifyNode** method. If the UDP message is successfully received by the node, and it responds to the message, a **vNode** object is returned from the method. This **vNode** object can then be cast to a **vNodeId** object and used in the constructor for the desired Server Object.

Note - A directed UDP message is also sent under the following conditions: If you call the getANode method on sNodeManager when you are using a TIBCO iProcess Objects Server that does not support the DLS message (which was added to support the TIBCO iProcess Objects Director), the method detects that it's an older server and sends a directed UDP message to the server instead of the DLS message. After receiving a reply from the server, the method will return a vANode object.

Specifying a UDP Port

By default, UDP messages are issued by the client on port 55666. The **sNodeManager** object contains the following methods for getting/setting the port on which UDP messages are issued:

- **getUDPPortNumbers** - This returns an array of the currently set UDP port(s).
- **setUDPPortNumbers** - This allows you to specify the port number(s) on which directed UDP messages will be sent out.

By default:

- TIBCO iProcess Objects Servers listen for UDP messages on port 55666; for information about how to specify the port on which TIBCO iProcess Objects Servers listen, see the **UDPServiceName** configuration parameter in the *TIBCO iProcess Objects Server Administrator's Guide.*
- TIBCO iProcess Objects Directors listen on port 28001; for information about specifying the port on which TIBCO iProcess Objects Directors listen, see the **UDP_SERVICE_NAME** process attribute in the *TIBCO iProcess Objects Director Administrator's Guide*.

It may be necessary to send out the directed UDP message on multiple ports when running more than one TIBCO iProcess Objects Server / Director on the same machine. Some implementations of the TCP/IP stack will only deliver the UDP message to one service. In this case, each TIBCO iProcess Objects Server / Director must be configured to use its own port. The client must then be configured to send the UDP message on each of those ports — the **setUDPPortNumbers** method allows for specifying multiple UDP ports.

Multiple Instances of the TIBCO iProcess Objects Server / Director

When running *multiple instances* of the TIBCO iProcess Objects Server / Director on a single machine, each instance must use a unique UDP port.

The vNode object returned by a TIBCO iProcess Objects Server / Director responding to a UDP message contains an *instance number* to identify that specific instance of the TIBCO iProcess Objects Server / Director. This number is available with the **getInstance** method on vNode. For TIBCO iProcess Objects Servers that don't support multiple instances, or when only a single instance is running on a machine, the getInstance method returns a 1.

- You can configure the UDP port on which a TIBCO iProcess Objects Server listens for UDP broadcasts by using the **UDPServiceName** configuration parameter. For more information about running multiple instances of the TIBCO iProcess Objects Server, see the *TIBCO iProcess Objects Server Administrator's Guide.*
- You can configure the UDP port on which a TIBCO iProcess Objects Director listens for UDP broadcasts by using the **UDP_SERVICE_NAME** process attribute. For more information about running multiple instances of the TIBCO iProcess Objects Director, see the *TIBCO iProcess Objects Director Administrator's Guide*.

Sending a UDP Broadcast

Sending a User Datagram Protocol (UDP) broadcast is done for the purpose of auto-discovering available nodes (TIBCO iProcess Objects Servers or TIBCO iProcess Objects Directors) on the LAN segment. The UDP broadcast requests that all TIBCO iProcess Objects Servers or TIBCO iProcess Objects Directors on the LAN segment return a message identifying themselves. The information they return can then be used to construct a Server Object and establish a user session with the TIBCO iProcess Objects Server, or to have a TIBCO iProcess Objects Director connect the client to a TIBCO iProcess Objects Server.

A UDP broadcast is issued on the LAN segment using the following two methods:

- **sNodeManager.getNodes** - A UDP broadcast is issued the *first* time you call this method and only if the **refresh** method (see below) had not been called prior to calling this method. The **getNodes** method returns a **vNode** object for each node that the Node Manager knows about.

  Subsequent calls to **getNodes** return a **vNode** object for each node that the Node Manager currently knows about — it does NOT issue another broadcast (use the **refresh** method to re-broadcast).
- **sNodeManager.refresh** - This refreshes the **sNodeManager's** cache of known nodes by issuing another UDP broadcast. After the re-broadcast, get the new list of known nodes by calling the **getNodes** method.

Setting the UDP Broadcast Interval

Occasionally, machines that are listening on the network miss the auto-discovery UDP broadcast. This is the nature of the UDP broadcast mechanism — there is no guarantee that available nodes will hear and respond to the broadcast. To compensate for this, you can specify the number of UDP broadcasts that are issued by using the *aPollCnt* parameter when constructing the **sNodeManager** object:

```
sNodeManager(short aPollCnt)
```

Broadcasts are issued once per second, with the total number of broadcasts equaling the value of *PollCnt*.

Note that *PollCnt* defaults to 5 if you use the **sNodeManager** constructor with no parameters.

Specifying a UDP Port

By default, UDP broadcasts are issued on port 55666. The **sNodeManager** object contains the following methods for getting/setting the port on which UDP broadcasts are issued:

- **getUDPPortNumbers** - This returns an array of the currently set UDP port(s).
- **setUDPPortNumbers** - This allows you to specify the port number(s) on which UDP broadcasts will be sent out.

By default:

- TIBCO iProcess Objects Servers listen for UDP broadcasts on port 55666; for information about how to specify the port on which TIBCO iProcess Objects Servers listen, see the **UDPServiceName** configuration parameter in the *TIBCO iProcess Objects Server Administrator's Guide*;
- TIBCO iProcess Objects Directors listen on port 28001; for information about specifying the port on which TIBCO iProcess Objects Directors listen, see the **UDP_SERVICE_NAME** process attribute in the *TIBCO iProcess Objects Director Administrator's Guide*.

It may be necessary to send out the UDP broadcast on multiple ports when running more than one TIBCO iProcess Objects Server / Director on the same machine. Some implementations of the TCP/IP stack will only deliver the UDP broadcast to one service. In this case, each TIBCO iProcess Objects Server / Director must be configured to use its own port. The client must then be configured to send the UDP broadcast on each of those ports — the **setUDPPortNumbers** method allows for specifying multiple UDP ports.

Multiple Instances of the TIBCO iProcess Objects Server / Director

When running *multiple instances* of the TIBCO iProcess Objects Server / Director on a single machine, each instance must use a unique UDP port.

The vNode object returned by a TIBCO iProcess Objects Server / Director responding to a UDP broadcast contains an *instance number* to identify that specific instance of the TIBCO iProcess Objects Server / Director. This number is available with the **getInstance** method on vNode. For TIBCO iProcess Objects Servers that don't support multiple instances, or when only a single instance is running on a machine, the getInstance method returns a 1.

- You can configure the UDP port on which a TIBCO iProcess Objects Server listens for UDP broadcasts by using the **UDPServiceName** configuration parameter. For more information about running multiple instances of the TIBCO iProcess Objects Server, and setting the **UDPServiceName** configuration parameter, see the *TIBCO iProcess Objects Server Administrator's Guide*.
- You can configure the UDP port on which a TIBCO iProcess Objects Director listens for UDP broadcasts by using the **UDP_SERVICE_NAME** process attribute. For more information about running multiple instances of the TIBCO iProcess Objects Director, and setting the **UDP_SERVICE_NAME** process attribute, see *TIBCO iProcess Objects Director Administrator's Guide*.

What if a Known Node is not Answering the UDP Broadcast?

There are a number of possible reasons this may be occurring:

- The most common reason is because the TIBCO iProcess Objects Server / Director is on the other side of a router (routers usually don't allow UDP broadcasts through) and didn't hear the broadcast. If this is the case, your options are:
  - Move the TIBCO iProcess Objects Server / Director to the other side of the router.
  - Create the **vNodeId** object. This, of course, requires that you know all of the pertinent information about the node. See *Constructing a vNodeId Object* on page 33.
  - Use the **verifyNode** method. A directed UDP message can go across a router, where the broadcast can't. See *Sending a Directed UDP Message* on page 34.
- The broadcast interval may be set too small. Try increasing the interval by using the *PollCnt* parameter when constructing the **sNodeManager** object. See *Setting the UDP Broadcast Interval* on page 35.
- It's also possible that the TIBCO iProcess Objects Server's / Director's service is not running. Ensure that the service is started on the machine on which it's installed.
- The wrong UDP port may be specified on the client. You can check this with the **getUDPPortNumbers** method. See *Specifying a UDP Port* on page 36.
- If there are multiple TIBCO iProcess Objects Servers / Directors running on a single machine, the UDP message may only be delivered to one of the TIBCO iProcess Objects Servers / Directors. To remedy this, each TIBCO iProcess Objects Server / Director must be configured to use its own UDP port. For information about how to specify the port on which TIBCO iProcess Objects Servers listen, see the **UDPServiceName** configuration parameter in the *TIBCO iProcess Objects Server Administrator's Guide*; for information about specifying the port on which TIBCO iProcess Objects Directors listen, see the **UDP_SERVICE_NAME** process attribute in the *TIBCO iProcess Objects Director Administrator's Guide*. To cause UDP broadcast go out on multiple ports, specify multiple ports using the **setUDPPortNumbers** method.

Configuring the TIBCO iProcess Objects Server TCP Port

Server Objects communicate with the TIBCO iProcess Objects Server via TCP/IP. This requires that a TCP port be configured on the TIBCO iProcess Objects Server. The TCP port on the TIBCO iProcess Objects Server can be configured either as **dynamic** (also called ephemeral) or **static**, depending on the method you are using to locate nodes on the network:

- **Dynamic** - This assignment (which is the default) causes the O/S to dynamically assign the TCP port number when the TIBCO iProcess Objects Server starts. Use this assignment if you are either issuing a UDP broadcast or a directed UDP to a specific node. (Note that starting and stopping the TIBCO iProcess Objects Server usually results in a different TCP port being assigned each time.)
- **Static** - This assignment causes the TCP port number to always remain the same for that server. Use this assignment if you are manually creating the **vNodeId** object. This is required because you must specify the TCP port number in a parameter when you construct the **vNodeId** object. Therefore, you must know the TCP port the server is going to be using.

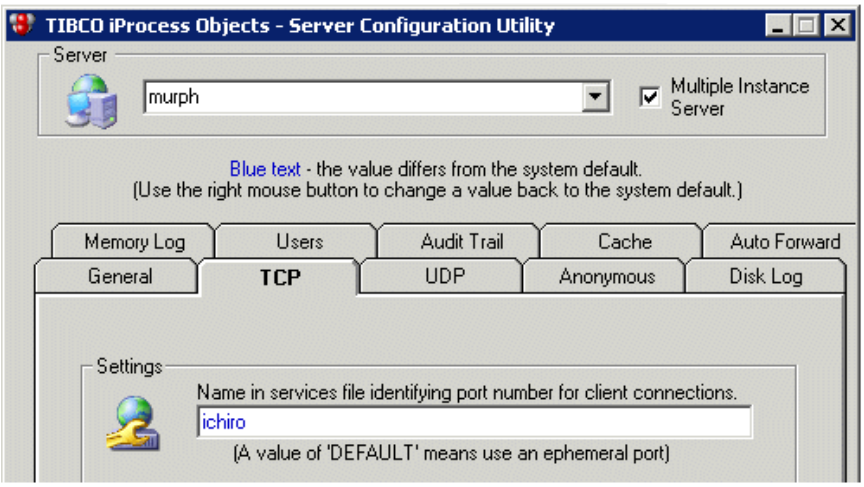
  Note that all TIBCO iProcess Objects Servers that want to make use of the TIBCO iProcess Objects Director must use a static TCP port. This allows the TIBCO iProcess Objects Director to be configured with those port numbers so it knows the TCP port number to use when establishing a connection between a Server Object and a TIBCO iProcess Objects Server. For more information, see the *TIBCO iProcess Objects Server TCP Port Configuration* section in the *TIBCO iProcess Objects Director Administrator's Guide*.

Note - For information about specifying the TCP ports on which the TIBCO iProcess Objects Servers listen when running multiple instances of the TIBCO iProcess Objects Server, see the TIBCO iProcess Objects Server Administrator's Guide.

Configuring the TCP Port on a Windows System

To configure the TCP port number on a TIBCO iProcess Objects Server running Windows, follow these steps:

1. Run the **TIBCO iProcess Objects Server Configuration Utility** control panel applet (for information about using this utility, see the *TIBCO iProcess Objects Server Administrator's Guide*) and click on the **TCP** tab (as shown below).



2. To configure the TCP port as **dynamic**, enter **DEFAULT** in the field in the TCP Port section, then click **OK**.

   To configure the TCP port as **static**, do one of the following:

   i. Enter the desired TCP port number in the field in the TCP Port section, then click **OK**, or

   ii. enter a "service name" in the field in the TCP Port section. This service name will be used to map to the TCP port number. If you use a service name, you must also edit the **%system-root%\system32\drivers\etc\services** file to add the service name and the desired TCP port number. The service name can be any name that is unique within the **services** file. The port number can be any number between 1024 - 65535 that is not already used in the **services** file. An example **services** file entry for a TCP port is shown below:

   ```
   ichiro   6666/tcp      # TCP port assignment
   ```

   After entering the service name in the field in the TCP Port section, click **OK**.

3. Stop, then restart the TIBCO iProcess Objects Server. For information about stopping and starting the TIBCO iProcess Objects Server, see the *TIBCO iProcess Objects Server Administrator's Guide*.

Configuring the TCP Port on a UNIX System

To configure the TCP port number on a TIBCO iProcess Objects Server running UNIX, follow these steps:

1. As the **root** user, open the **$**SWDIR**/seo/data/swentobjsv.cfg** file with a text editor and find the **TCPServiceName** entry (where $SWDIR is the directory in which the TIBCO iProcess Engine is installed).
2. Ensure the # symbol is removed to enable the **TCPServiceName** entry.
3. To configure the TCP port as **dynamic**, set **TCPServiceName** equal to **DEFAULT**.

   To configure the TCP port as **static**, do one of the following:

   i. Set **TCPServiceName** to the desired TCP port number, or

   ii. Set **TCPServiceName** to a "service name." This service name will be used to map to the TCP port number. If you use a service name, you must also edit the **/etc/services** file to add the service name and the desired TCP port number. The service name can be any name that is unique within the **services** file. The port number can be any number between 1024 - 65535 that is not already used in the **services** file. An example **services** file entry for a TCP port is shown below:

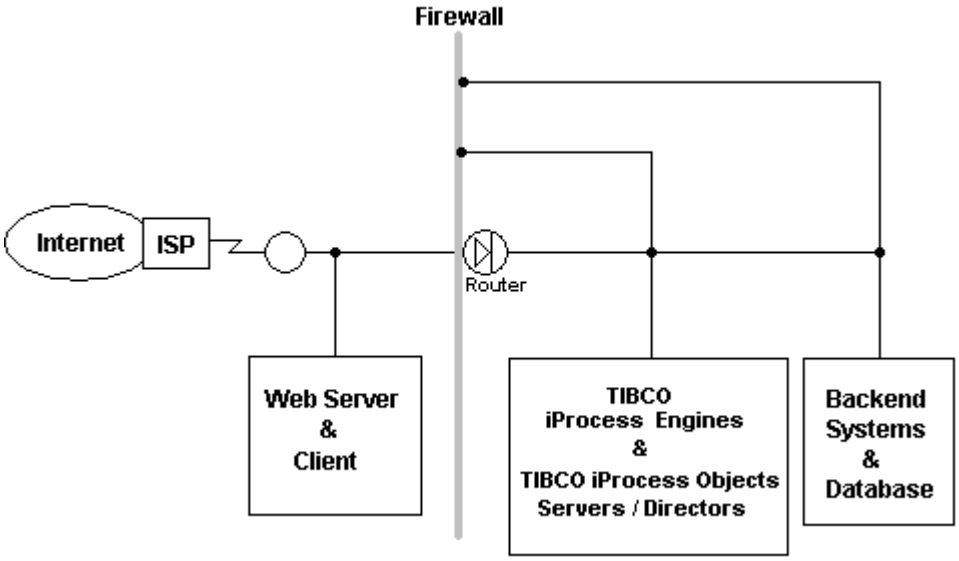
   ```
   ichiro   6666/tcp      # TCP port assignment
   ```

4. Save the edited **$**SWDIR**/seo/data/swentobjsv.cfg** file.
5. Stop, then restart the TIBCO iProcess Objects Server. For information about stopping and starting the TIBCO iProcess Objects Server, see the *TIBCO iProcess Objects Server Administrator's Guide*.

Using TIBCO iProcess Server Objects Through a Firewall?

TIBCO iProcess Server Objects can be used through a firewall. To do so, you must:

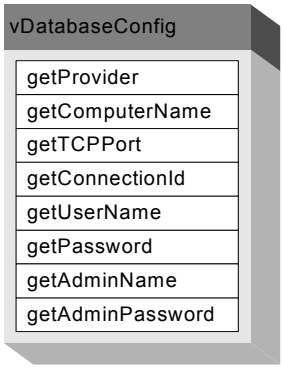
- Assign the TIBCO iProcess Objects Server or TIBCO iProcess Objects Director a static TCP port (for information about assigning a static TCP port for the TIBCO iProcess Objects Server, see *Configuring the TIBCO iProcess Objects Server TCP Port* on page 38; for information about assigning a static TCP port for the TIBCO iProcess Objects Director, see the **TCP_SERVICE_NAME** process attribute in the *TIBCO iProcess Objects Director Administrator's Guide*).
- Manually construct a **vNodeId** object that represents the desired TIBCO iProcess Objects Server or TIBCO iProcess Objects Director (the vNodeId object is used in the construction of the desired Server Object).
- On the firewall, open up the TCP port used by the TIBCO iProcess Objects Server or TIBCO iProcess Objects Director to communicate with the client.



Database Configuration

Note - This functionality is not available in the EJB interface.

The **vDatabaseConfig** object exposes database configuration information on the TIBCO iProcess Engine. This object can be obtained by calling the **getDatabaseConfig** method on the **sIPEConfig** object. The vDatabaseConfig object provides the following methods to obtain database configuration information:



- **getProvider** - The name of the database provider, which defaults to one of the following, depending on the type of database:
  - ORACLE
  - SQL_SERVER
  - DB2
- **getComputerName** - The machine name on which the database is installed.
- **getTCPPort** - The TCP port number used to connect to the database.
- **getConnectionId** - The database ID, which defaults to the following, depending on the database provider:
  - Oracle - Either the Oracle SID (for direct connections) or the TNS (Transparent Network Substrate) connection name (for TNS connections).
  - SQL Server - ODBC connection name.
  - DB2 - DB2 alias name.
- **getUserName** - The iProcess Engine database foreground user name, which defaults to "swuser".
- **getPassword** - The iProcess Engine database foreground user's password.
- **getAdminName** - The iProcess Engine IPEADMIN user name. (Note - The IPEADMIN user is the iProcess Engine system administrator. This user is designated when the TIBCO iProcess Engine is installed. It defaults to the user installing the iProcess Engine, but can be specified as any iProcess user.)
- **getAdminPassword** - The iProcess Engine iPEADMIN user's password.

Database Configuration Access

The TIBCO iProcess Objects Server contains a configuration parameter (**DBConnectionAccess**) that is used to specify whether or not to allow access to the database configuration information. It can be set to:

- allow access to all users,
- allow access to only System Administrators (for information about System Administrator authority, see *User Authority* on page 190),
- disable access so database configuration information is not available.

For more information about this configuration parameter, see the *TIBCO iProcess Objects Server Administrator's Guide*.

Activity Publication

The TIBCO iProcess Engine can be enabled to publish iProcess Engine activity information to external applications. Any activity (i.e., anything that generates an audit trail message, for example, a case start or deadline expiration) can be monitored and enabled for publication. This can be configured per procedure or for all procedures, depending on your requirements. This means that an external application can monitor important business events during the processing of cases.

The Background process identifies if activity publication has been enabled for an activity as it is being processed. If activity publication has been enabled, the Background process outputs Java Message Service (JMS) messages containing details of the published activities. These JMS messages are sent to the IAP JMS Library (Introspection and Activity Publication JMS Library).

The IAP JMS library sends the JMS messages to a specified JMS topic or queue name, from which the external application can read the JMS messages.

For more information about introspection and activity publication, see the *Monitoring Activities* chapter in the *TIBCO iProcess Engine Administrator's Guide*.

Activity Publication Access

The TIBCO iProcess Objects Server contains a configuration parameter (**IAPConfigAccess**) that is used to specify whether or not to allow access to activity publication configuration information. It can be set to:

- allow access to all users,
- allow access to only System Administrators (for information about System Administrator authority, see *User Authority* on page 190),
- disable access so activity publication configuration information is not available.

For more information about this configuration parameter, see the *TIBCO iProcess Objects Server Administrator's Guide*.

Configuring Activity Publication

Configuration information for activity publication is stored in the database. To configure activity publication information for your iProcess Engine, you must do the following:

- Generate your activity publication configuration information as XML in the form of a **Message Event Request (MER)** message. You can do this using any available XML tool. The MER message must conform to the **SWMonitorList.xsd** schema (which is written to the *SWDIR***schemas** directory when the iProcess Engine is installed).
- Once you have generated an MER message according to your requirements, there are two ways to update the activity publication configuration information in the database with the activity publication configuration information in the new MER message:
  - using the **swutil IMPMONITOR** command — for information about this command, see the *Activity Monitoring* chapter in the *TIBCO iProcess swutil and swbatch Reference Guide*.
  - using the **sIPEConfig** object — use of this object is described below.

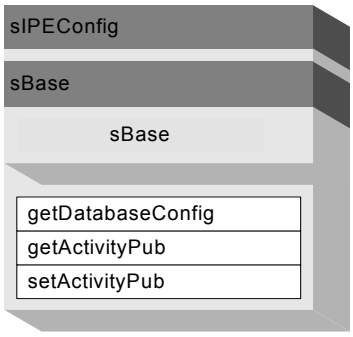
- Using the **Manage Monitor Event Request** tool in the TIBCO iProcess Workspace (Browser) application. This tool provides an interface that allows you to create a new MER message, or edit an existing MER on the TIBCO iProcess Engine.

  The TIBCO iProcess Workspace (Browser) also provides an IPC Tools method (**ipcManageMER**) and a WCC Tools method (**manageMER**) to access the Manage Monitor Event Request interface from custom applications.

  For more information, see the TIBCO iProcess Workspace (Browser) documentation.

Using the sIPEConfig Object

The **sIPEConfig** object contains two methods that allow you to either get or set the activity publication configuration for the iProcess Engine to which you connected when the sIPEConfig object was constructed. It contains the following methods:



- **getActivityPub** - This method returns an XML string containing the activity publication configuration for the engine. You can specify that you want the configuration information for all procedures or a specific procedure.
- **setActivityPub** - This method sets the activity publication configuration based on the MER message (XML string) sent in the method call. The XML string must conform to the **SWMonitorList.xsd** schema. An example MER message is shown in the next subsection.

Configuration Example

An example MER message (which conforms to the **SWMonitorList.xsd** schema) generated to configure activity publication for a procedure called BANK01 is shown below. The table summarizes the configuration generated in the MER message. The table shows:

- the activities to be monitored
- the activity number (this relates to the audit trail number — see **SWAuditActionType** on page 70 — an activity number of -1 means monitor all activities)
- the steps on which activities are to be monitored (ALL means all steps in the specified procedure(s))
- the field data to be published when the activity occurs.

Activity	Activity Number	Step Name	Field Name
Case start	0	ALL	ACCNO
			SURNAME
			LOAN_AMOUNT
All	-1	MTGACC01	ACCNO
			SURNAME
			LOAN_AMOUNT
Deadline expired	3	ALL	ACCNO

Activity	Activity Number	Step Name	Field Name
Case terminated	9	ALL	SURNAME LOAN_AMOUNT DEAD_REASON DEAD_DATE ACCNO SURNAME LOAN_AMOUNT DECISION CLOSED_DATE

The MER message generated to represent this configuration is shown below:

```
<ProcedureMonitor xmlns="http://bpm.tibco.com/2004/IAP/MER"
xmlns:ns2="http://bpm.tibco.com/2004/IAP/1.0/SWTypes"
xmlns:ns1="http://bpm.tibco.com/2004/IAP/1.0/procedureProperties"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation
  ="http://bpm.tibco.com/2004/IAP/MER:\Projects\1439\Docs\schemas\SWMonitorList.xsd">
    <SchemaVersion>001</SchemaVersion>
    <MessageType>MER</MessageType>
    <FullImport>true</FullImport>

    <MonitorDetail>
        <Procedure Name="BANK01">
          <NodeName>SWNOD1</NodeName>
        </Procedure>
        <GlobalFieldList>
          <Field Name="REQUEST_ID"/>
          <Field Name="REQUEST_DATE"/>
          <Field Name="REQUEST_STS"/>
        </GlobalFieldList>
        <MonitorList>
          <Monitor>
            <ActivityList>
              <Activity Num="0"/>
            </ActivityList>
            <StepList>
              <Step Name="$ALL$"/>
            </StepList>
            <FieldList>
              <Field Name="ACCNO"/>
              <Field Name="SURNAME"/>
              <Field Name="LOAN_AMOUNT"/>
            </FieldList>
          </Monitor>
          <Monitor>
            <ActivityList>
```

```
            <Activity Num="-1"/>
          </ActivityList>
          <StepList>
            <Step Name="MTGACC01"/>
          </StepList>
          <FieldList>
            <Field Name="ACCNO"/>
            <Field Name="SURNAME"/>
            <Field Name="LOAN_AMOUNT"/>
          </FieldList>
        </Monitor>
        <Monitor>
          <ActivityList>
            <Activity Num="3"/>
          </ActivityList>
          <StepList>
            <Step Name="MTGACC01"/>
          </StepList>
          <FieldList>
            <Field Name="ACCNO"/>
            <Field Name="SURNAME"/>
            <Field Name="LOAN_AMOUNT"/>
            <Field Name="DEAD_REASON"/>
            <Field Name="DEAD_DATE"/>
          </FieldList>
        </Monitor>
        <Monitor>
          <ActivityList>
            <Activity Num="9"/>
          </ActivityList>
          <StepList>
            <Step Name="$ALL$"/>
          </StepList>
          <FieldList>
            <Field Name="ACCNO"/>
            <Field Name="SURNAME"/>
            <Field Name="LOAN_AMOUNT"/>
            <Field Name="DECISION"/>
            <Field Name="CLOSE_DATE"/>
          </FieldList>
        </Monitor>
      </MonitorList>
    </MonitorDetail>
</ProcedureMonitor>
```

6

Procedures

Introduction

A business process that is automated is referred to as a "procedure." Procedures are defined with a tool called "TIBCO Business Studio". A procedure consists of a number of "steps," including manual steps (which require user action), automatic steps (which are executed automatically by the server), and condition steps (which branch based on the result of a condition). An example of a simple procedure is shown below.

Managing Procedures

The **sProcManager** Server Object is used to manage procedures. This object allows you to retrieve Value Objects from the server that represent procedures, as well as the components that make up procedures (fields, steps, and forms).

For information about accessing instances of procedures (e.g., starting a case of a procedure), see "Case Management" on page 61.

For information about accessing procedure definition information, see the following subsections.

Increasing amounts of information about a particular procedure is available in the **vProcId**, **vProc**, and **vAProc** Value Objects. These objects can be retrieved from the server by calling the **getProcIds**, **getProcs**, and **getAProcs** methods, on **sProcManager**. These objects provide access to information such as the procedure name and number, how many active and closed cases of that procedure exist, etc. Notice that this is mostly "runtime" (dynamic) data.

Procedure definitions are represented by the **vProcDef** object. The vProcDef object contains methods that provide procedure definition information, such as whether the procedure is defined as a main or sub-procedure (isSubProc), who the owner is (getOwner), the steps that are defined in the procedure (getStepIds), etc. Notice that this is mostly "configuration" (static) data.

Procedure definitions can be obtained by calling the **getProcDefs** method on **sProcManager**. This method returns an array of **vProcDef** objects. You can specify which procedure definitions you want by including a procedure tag in the method call. All procedure definitions are returned unless specific procedure tags are passed as a parameter.

When retrieving vProcDef objects, you can specify how much dependent data (steps, audit data, etc.) you want included with the procedure definitions. This is done with the **vProcDefContent** object (see "Retrieving Dependent Objects" on page 160 for information about using the "content objects").

Procedure Version Control

Procedure version control provides the ability to create and track multiple versions of procedures. This allows you to develop and test a modified procedure while a live version is still in use. It also allows you to revert to a previous version if you need to. (*Note - Procedure version control is only supported on TIBCO iProcess Objects Servers versions i10 or newer.)*

When a procedure is created using TIBCO Business Studio, it is given a unique version number in the form:

<MajorVersion#>.<MinorVersion#>

For example, 1.0, 1.1, 1.2, 2.1, and so on.

There can be many versions of a particular procedure on your system at one time. All versions are saved until you explicitly delete them.

Procedure Status

Each version of a procedure also has a procedure status associated with it. The status dictates how the procedure can be used.

- **swReleased -** A procedure with this status can be used in live production. Cases can be started, with work items being processed to user's work queues.
- **swUnreleased -** New procedures default to a status of swUnreleased. Work items from cases of procedures with a status of swUnreleased go to a "test" work queue for the user or group who is the addressee of the step. This allows the new procedure to be tested/evaluated prior to releasing it.
- **swModel -** This is the status a released procedure has after being imported. This status allows new versions of a procedure to be imported without overwriting an existing released or unreleased version. Work items from cases of procedures with a status of swModel go to a "test" work queue for the user or group who is the addressee of the step. This allows the new version to be tested/evaluated prior to adopting it on the target system.
- **swWithdrawn -** Procedures with this status are no longer used in a production environment. Cases cannot be started against a withdrawn procedure. When a procedure is given a status of withdrawn, existing cases of the procedure are run to completion.
- **swIncomplete -** A procedure with this status cannot be run because it has required information missing — for example, a step without an addressee, or a step without a connection from a previous step. This procedure status is not supported in TIBCO iProcess Server Objects (Java), i.e., procedures with this status are not returned by the TIBCO iProcess Objects Server.
- **swWithdrawnIncomplete -** An incomplete procedure that has been withdrawn. This procedure status is not supported in TIBCO iProcess Server Objects (Java), i.e., procedures with this status are not returned by the TIBCO iProcess Objects Server.

Notice that for any particular procedure, there can only be:

- **one** swReleased version,
- **one** swModel version,
- **one** swUnreleased version, and
- **any number** of swWithdrawn versions.

You can determine a procedure's status by calling the **vProc.getStatus** method. They are enumerated in **SWProcStatusType**.

If multiple versions of a procedure exist, one of those versions is considered the "current" version. The "current" version is defined as the version with the highest status (**vProc.getStatus**), according to the following status hierarchy:

swReleased -> swUnreleased -> swModel -> swWithdrawn (most recent)

For example, if a particular procedure has a version with a status of swReleased, that is the "current" version. If the procedure doesn't have a version with a status of swReleased, but has one with a status of swUnreleased, that is the "current" version, and so on.

Some TIBCO iProcess Server Objects methods act upon or list only the "current" version of a procedure (e.g., the getProcs method only returns the "current" version of each procedure).

Accessing the Procedure Version Number

To allow access to the procedure version number in the TIBCO iProcess Server Objects, the following methods are available on the **vProcId** objects:

- **getMajorVersion** - This returns an integer indicating the <*MajorVersion#*> portion of the procedure's version number.
- **getMinorVersion** - This returns an integer indicating the <*MinorVersion#*> portion of the procedure's version number.

These methods tell you the version number of that procedure definition.

Procedure Version Number in the Audit Trail

When entries are written to the audit trail, the procedure version number is part of the entry. This is done because the version number may change mid-case, i.e., the case may be migrated to a new version of the procedure, resulting in steps (work items) of a case having different version numbers. (When a new version is created in TIBCO Business Studio, it asks you if you want existing cases of that procedure "migrated" to the new version, or whether they should be completed under the old version.)

The **vAuditStep** object contains the following methods to determine the procedure version number when the action represented by the vAuditStep object was performed:

- **getProcMajorVersion** - This returns an integer indicating the <*MajorVersion#*> portion of the procedure's version number when the audit action was performed.
- **getProcMinorVersion** - This returns an integer indicating the <*MinorVersion#*> portion of the procedure's version number when the audit action was performed.

Procedure Version Details

The **vAProc** object contains a number of methods that provide details about the version of the procedure:

- **getDateReleased** - This returns the date and time the procedure was released. If it has not been released, this method returns 12/31/3000 11:15:00 PM.
- **getDateCreated** - This returns the date and time this version of the procedure was created.
- **getDateModified** - This returns the date and time this version of the procedure was last modified. If this version of the procedure has not been modified, it returns 12/31/3000 11:15:00 PM.
- **getDateWithdrawn** - This returns the date and time this version of the procedure was withdrawn. If this version of the procedure has not been withdrawn, this returns 12/31/3000 11:15:00 PM.
- **getLastUpdateUser** - This returns the name of the user who last updated this version of the procedure.
- **getVersionComment** - This returns the comment that was entered by the user who last updated the procedure.

Listing Versions of a Procedure

If you are operating with a TIBCO iProcess Objects Server that supports procedure versions (version i10 or newer), the **getProcs** method will return a list containing the "current" version of each procedure on the node. See "Procedure Status" on page 48 for information about "current" versions. (With earlier versions of the TIBCO iProcess Objects Server, the getProcs method returned all procedures, regardless of their Status (except swIncomplete and swWithdrawnIncomplete, which are not supported by TIBCO iProcess Server Objects).)

With procedure version control, you can also obtain a list of ALL versions of a specified procedure using the following method:

- **sProcManager.getProcVersions** - This method returns a list of **vAProc** objects, each representing a specific version of the procedure. This allows you to list all versions of the procedure, rather than just the current version.

Accessing a Specific Procedure Version

There are a number of methods that require a "procedure tag" (*aProcTag* argument) to identify the procedure. This tag contains a version number component that identifies the specific version of the procedure, as there could be multiple versions of the procedure.

To accommodate the version number component in the tag, the **makeTag** method on **vProcId** requires that the version number be included in the method call:

```
String makeTag(String aNode,
               String aProcName,
               int aMajorVersion,
               int aMinorVersion)
```

Note - You can specify the "current" version of the procedure by passing -1 in the aMajorVersion and aMinorVersion parameters.

Procedure Audit Trails

When procedure definitions are modified in TIBCO Business Studio, information about the modification is written to an audit trail. This procedure audit trail information is available using the following method on **vProcDef**:

- **getProcAudits** - This method returns an array of **vProcAudit** objects, each representing a specific modification to the procedure definition.

When you get the **vProcDef** objects from the server, you must set the *isWithAuditData* parameter on the procedure definition content object (**vProcDefContent**) to True so that the procedure audit data (**vProcAudit** objects) is also returned from the server.

The **vProcAudit** object contains the following methods, which provide information about the procedure modification:

- **getAction** - Describes the modification made to the procedure — these actions are defined in the enumeration type **SWProcAuditActionType**.
- **getComment** - User comments concerning modifications made to the procedure.
- **getDate** - The date and time the modification occurred.
- **getProcMajorVersion** - The "major" portion of the procedure version number when the procedure definition is modified.
- **getProcMinorVersion** - The "minor" portion of the procedure version number when the procedure definition is modified.
- **getUser** - The name of the user who made the modifications.

Sub-Procedures

Sub-procedures provide the ability for a case of one procedure to start a case of another procedure as one of its steps. When the case of the child procedure has completed, the actions of the sub-procedure call step in the parent case are processed just as for a normal step.

When a sub-procedure is started, flow is halted along that particular path of the calling procedure until the sub-procedure has completed.

When a procedure is defined in TIBCO Business Studio, you specify that it is either a "main" procedure or a "sub-procedure". A main procedure is started directly with the **startCase** method. An instance (sub-case) of a sub-procedure can only be started by one of the follow types of steps in a procedure:

- **Sub-procedure call step** - This type of call step (also called a "static" sub-procedure call step) starts a single case of a sub-procedure. When the sub-procedure call step is defined in TIBCO Business Studio, you specify the sub-procedure that will be started when the process flow reaches the sub-procedure call step. See "Sub-Procedure Call Steps" on page 52 for more information.
- **Dynamic sub-procedure call step** - This type of call step allows you to dynamically start one or more sub-procedures. When the dynamic sub-procedure call step is defined in TIBCO Business Studio, rather than specifying the sub-procedures to start, you specify the name of an array field. At run-time, the customer application will write the names of sub-procedures into the elements of the array field. When the process flow reaches the dynamic sub-procedure call step, the sub-procedures specified in the elements of the array field are started. See "Dynamic Sub-Procedure Call Steps" on page 53 for more information.
- **Graft steps** - This type of call step is similar to a dynamic sub-procedure call step in that it allows you to dynamically start one or more sub-procedures. The way in which it differs is that it allows the application to start multiple sub-procedures as part of a "task". A task can also involve starting external processes, and you can start multiple tasks. See "Using Graft Steps" on page 91 for more information.

Note that sub-procedures can be many levels deep, i.e., a sub-procedure can also contain a sub-procedure call step, and the sub-procedure started by that call step can contain a graft step, and so on.

Also, a sub-procedure case cannot be directly closed or purged — although the main case from which the sub-procedure was called can be closed or purged.

TIBCO iProcess Server Objects provide read-only access to the properties associated with sub-procedures. Therefore, access is provided to the definition of sub-procedures, but they may only be defined and modified in TIBCO Business Studio, and not through TIBCO iProcess Server Objects.

Sub-Procedure Call Steps

When a sub-procedure call step (also called a "static" sub-procedure call step) is defined in TIBCO Business Studio, you specify the name of the sub-procedure to start when the process flow reaches the sub-procedure call step. This name is available with the **getSubProcName** method on the **vSubProcCallStep** object that represents the sub-procedure call step.

When a sub-procedure call step is defined, you can also specify a start step other than the default start step. The name of this alternative start step is available with the **getSubProcStartStep** method on the vSubProcCallStep object that represents the sub-procedure call step. If a start step other than the default start step was not specified, the getSubProcStartStep method will return an empty string.

Accessing Sub-Procedure Call Step Definitions

To access the definition of a sub-procedure call step in a procedure, call **sProcManager.getSteps** or **vProcDef.getStepIds** to get an array of **vStepId** objects, one for each step in the procedure. Use **getType** to identify the sub-procedure call step (**SWStepType** = swSubProcCall), then cast it to a **vSubProcCallStep** object.

Sub-Procedure Start Precedence

When you start a main procedure with the **startCase** method, you can also specify an *aSubProcPrecedence* parameter that allows you to specify the "precedence" of sub-procedure statuses (released, unreleased, or model) that are launched from the main procedure. In other words, you are telling it which status to look for first, second, then third. For example, you can specify that it look for unreleased, then model, then released statuses. The default is to start only released statuses. See "Sub-Procedure Precedence" on page 64 for more information.

Dynamic Sub-Procedure Call Steps

A "static" sub-procedure call always starts a case of the same sub-procedure, whereas a "dynamic" sub-procedure call step allows you to specify at *run-time* the names of one or more sub-procedures to start. When a dynamic sub-procedure call step is processed (as an action of another step), all of the sub-procedures specified are started. The TIBCO iProcess Engine will keep track of all of the sub-procedures that were started — when they have all completed, it will process the dynamic sub-procedure call step's release actions.

Dynamic sub-procedure call steps work in the same way as static sub-procedure calls, with the following exceptions:

- **Sub-Procedures to Start** - When a dynamic sub-procedure call step is defined in TIBCO Business Studio, you do not specify the sub-procedures that will be started when the step is processed. Instead, you specify an "array field", which can contain multiple elements that are accessed by index. At run-time, the customer application must write the names of the sub-procedure to start in the elements of the array field. When the step is processed, those sub-procedures are started.

  The name of the array field containing the names of the sub-procedures to start is available with the **vDynamicSubProcCallStep.getSubProcNameFld** method.

  If no elements of the "sub-procedures to start" array field are assigned when the process flow reaches the dynamic sub-procedure call step, the step is immediately released and its release actions are performed.

  See "Array Fields" on page 172 for information about how array fields are used with dynamic sub-procedure call steps.
- **Sub-Procedure Start Steps** - When a dynamic sub-procedure call step is defined in TIBCO Business Studio, you can specify an array field whose elements will contain alternative start steps at which each sub-procedure will be started. At run-time, the application can write the names of the start steps into the elements of the array field. The elements in the "start step" array field must correspond to the sub-procedures in the array field identified by the getSubProcName method.

  The name of the array field containing the names of the start steps on which to start is available with the **vDynamicSubProcCallStep.getSubProcStartStepFld** method.

If an element that corresponds to one of the sub-procedures is empty when the step is processed, that sub-procedure will start on its default start step.

See "Array Fields" on page 172 for information about how array fields are used with dynamic sub-procedure calls.

- **Return Status** - When a dynamic sub-procedure call step is defined in TIBCO Business Studio, you can specify an array field, whose elements will contain a return status for each corresponding sub-procedure that is started by the dynamic sub-procedure call step. The name of the array field containing the return statuses is accessible with the **getReturnStatusFld** method on the **vDynamicSubProcCallStep**. The elements of the array field will return an **SWSubProcStatusType** enumeration, identifying each sub-procedure's current status (whether it's started, completed, encountered an error, etc.), as follows:

SWSubProcStatusType	
swNoAttempt	0
swStarted	1
swCompleted	2
swErrSubProc	-1
swErrTemplate	-2
swErrInTemplateVer	-3
swErrOutTemplateVer	-4

The return status for each sub-procedure that is started by the dynamic sub-procedure call step is also available with the **getReturnStatus** method on the **vSubProcCase** object that represents the sub-procedure that was started by the dynamic sub-procedure call step.

- **Error Processing** - Dynamic sub-procedure call step definitions provide options that allow the definer to specify how continued processing will occur if an error is encountered during processing. These options are accessible with the following methods on **vDynamicSubProcCallStep**:
  - **isHaltOnSubProc** - Returns True if processing should be halted when the "sub-procedures to start" array field contains elements that specify non-existent sub-procedures.
  - **isHaltOnTemplate** - Returns True if processing should be halted when the "sub-procedures to start" array field contains elements that specify sub-procedures that do not use the same parameter template. (Parameter templates are used when defining procedures to ensure that the same input and output parameters are used when starting multiple sub-procedures from a dynamic sub-procedure call step — for information about parameter templates, see the *TIBCO Business Studio* documentation.)
  - **isHaltOnTemplateVer** - Returns True if processing should be halted when the "sub-procedures to start" array field contains elements that specify sub-procedures that do not use the same version of parameter template.

These options for halting processing on specific error conditions have the following affects:

Errors during initial processing (when the dynamic sub-procedure step is processed as an action of another step):

- If an error is encountered and the step is defined to halt:

- The message that resulted in the error will be retried the number of times specified in the TIBCO iProcess Engine. (This is specified with a background attribute: IQL_RETRY_COUNT = the number of times the message will be retried; IQL_RETRY_DELAY = the number of seconds between retries.) If the message retries do not result in a successful initial processing, the following apply:
  - Processing of the entire step is halted at this point — it will always be left "waiting" for the sub-case that's in error to be completed.
  - All sub-procedures that have been started from the step are rolled back.
  - An SW_ERROR message is logged stating the reason for the failure.
  - An appropriate entry is written to the audit trail for the parent case.

- If an error is encountered and the step is defined to NOT halt:
  - The other valid sub-procedures specified in the getSubProcNameFld array field will be started as usual.
  - An SW_WARN message is logged stating the reason for the failure.
  - An appropriate entry is written to the audit trail for the parent case.

Errors during completion processing of one of the sub-cases:

- If an error is encountered and the step is defined to halt:
  - The message that resulted in the error will be retried the number of times specified in the TIBCO iProcess Engine. (This is specified with a background attribute: IQL_RETRY_COUNT = the number of times the message will be retried; IQL_RETRY_DELAY = the number of seconds between retries.) If the message retries do not result in a successful completion processing, the following apply:
    - Processing of the entire step is halted at this point — it will always be left "waiting" for the sub-case that's in error to be completed.
    - The "sub-case completed" transaction for the sub-case in error is aborted — this does not cause transactions from other valid sub-case completions to be aborted.
    - An SW_ERROR message is logged stating the reason for the failure.
    - An appropriate entry is written to the audit trail for the parent case.

- If an error is encountered and the step is defined to NOT halt:
  - The "sub-case completed" transaction for the sub-case in error is ignored (including returned output parameter data).
  - The status of the sub-case is set to "complete" so that the step can be released when all other sub-cases complete.
  - An SW_WARN message is logged stating the reason for the failure.
  - An appropriate entry is written to the audit trail for the parent case.

Note that if none of the “halt on” selections are selected in TIBCO Business Studio, and one of the error conditions are encountered (e.g., sub-procedures using different templates), the process will continue, which could possibly result in errors in case data.

Accessing Dynamic Sub-Procedure Call Step Definitions

To access the definition of a dynamic sub-procedure call step in a procedure, call **sProcManager.getSteps** or **vProcDef.getStepIds** to get an array **vStepId** objects, one for each step in the procedure. Use **getType** to identify the dynamic sub-procedure call step (**SWStepType** = swDynamicSubProcCall), then cast it to a **vDynamicSubProcCallStep** object.

Passing Data between a Main and Sub-Procedure

Field values can be exchanged between main procedures and sub-procedures at the time a sub-procedure is started, and again when it completes. The list of fields, passed to and from the child case, is specified when the sub-procedure call step or dynamic sub-procedure call step is defined in TIBCO Business Studio.

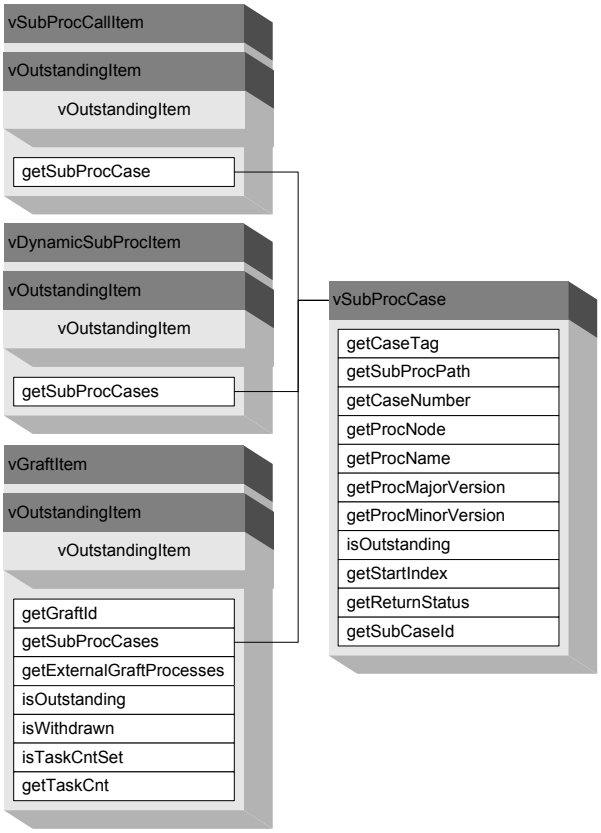
The **vSubProcCallStep** and **vDynamicSubProcCallStep** objects contain the following methods to access the list of fields that are passed to and from the child case (if such fields have been specified in the call step):

- **getInFromFldNames** - Returns an array of source fields, from the parent case, whose values are passed to the sub-case when it starts.
- **getInToFldNames** - Returns an array of destination fields, in the sub-case, that receive values at sub-case start.
- **getOutFromFldNames** - Returns an array of source fields, from the sub-case, whose values are passed to the parent case when the sub-case terminates.
- **getOutToFldNames** - Returns an array of destination fields, in the parent case, that receive values from the sub-case when it terminates.

The Sub-Case Object

Every sub-procedure that is started by a sub-procedure call step, dynamic sub-procedure call step, or graft step results in a **vSubProcCase** object. This object represents the live case for the sub-procedure that was started.

On an outstanding sub-procedure call step (**vSubProcCallItem** object), calling the **getSubProcCase** method returns the vSubProcCase object that represents the single sub-procedure that was started when the process flow reached the sub-procedure call step.



On an outstanding dynamic sub-procedure call step (**vDynamicSubProcItem** object), calling the **getSubProcCases** method returns an array of vSubProcCase objects, one for each sub-procedure that was started when the process flow reached the dynamic sub-procedure call step. Note that getSubProcCases will return vSubProcCase objects for *all* of the sub-procedures that are started for the dynamic sub-procedure call step, whether they have completed or not. To determine if a particular sub-procedure is still "outstanding" (has not completed yet), call the **isOutstanding** method on the vSubProcCase object that represents the sub-procedure in question.

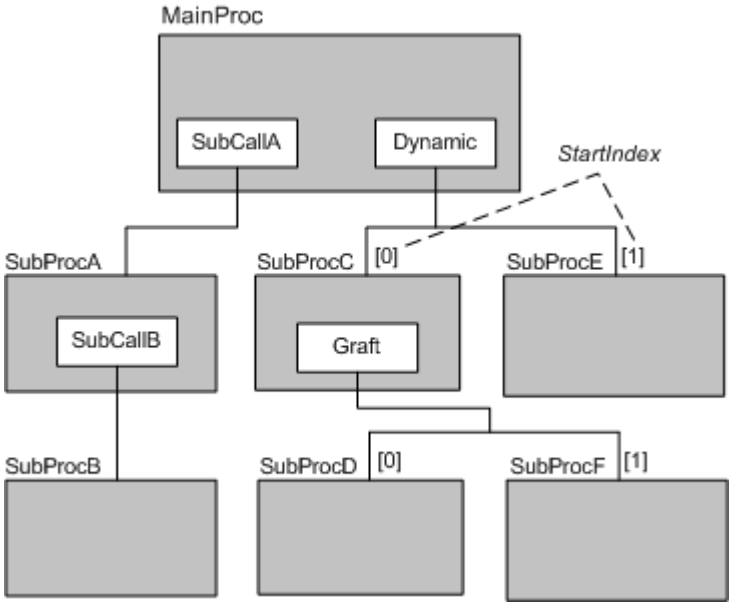
On an outstanding graft step (**vGraftItem** object), calling the **getSubProcCases** method returns an array of vSubProcCase objects, one for each sub-procedure that was started when the **startGraftTask** method was called for the graft step. Note that getSubProcCases will return vSubProcCase objects for *all* of the sub-procedures that are started for the graft step, whether they have completed or not. To determine if a particular sub-procedure is still "outstanding" (has not completed yet), call the **isOutstanding** method on the vSubProcCase object that represents the sub-procedure in question.

SubProcPath to a Sub-Case

The "SubProcPath" is a string that provides the path from the main procedure to an outstanding sub-procedure in the case family. This can be used in the *WithdrawList* parameter when calling the **jumpTo** method (withdrawing an outstanding sub-procedure causes all outstanding items in that sub-procedure to be withdrawn — see "Withdraw Outstanding Items / Jump To New Steps" on page 81 for more information).

Note - You can also determine the "ProcPath" from the main procedure to any outstanding item/step in the case family — see "ProcPath to Outstanding Items" on page 83 *for more information.*

The **getSubProcPath** method on **vSubProcCase** returns the SubProcPath to the sub-procedure represented by the vSubProcCase object. The illustration below shows example SubProcPath strings returned by the getSubProcPath method for a variety of sub-procedures.



Sub-Procedure	SubProcPath
SubProcA	“SubCallA”
SubProcB	“SubCallA|SubCallB”
SubProcC	“Dynamic[0]”
SubProcD	“Dynamic[0]|Graft[0]”
SubProcE	“Dynamic[1]”
SubProcF	“Dynamic[0]|Graft[1]”

If the sub-procedure is started by a sub-procedure call step that is in the main procedure, the SubProc-Path will simply consist of the name of the sub-procedure call step (see SubProcA in the example).

If the sub-procedure was started by a sub-procedure call step located in another sub-procedure, the SubProcPath string will consist of the name of each sub-procedure call step leading to the sub-procedure, each separated by a vertical bar (see SubProcB in the example).

If the case family contains dynamic sub-procedure call steps or graft steps that start multiple sub-procedures (see the Dynamic and Graft steps in the example), the name of the dynamic sub-procedure call step or graft step in the SubProcPath will include a **StartIndex** in square brackets. The StartIndex (which is zero based) indicates the sequential order in which the sub-procedure was started by the engine for that dynamic sub-procedure call step or graft step.

In addition to appearing in the SubProcPath as illustrated above, you can also determine the StartIndex for any particular sub-procedure that was started by a dynamic sub-procedure call step or graft step by calling the **getStartIndex** method on the **vSubProcCase** object that represents that sub-procedure.

The StartIndex is not applicable to sub-procedures that are started from sub-procedure call steps. If the sub-procedure was started from a sub-procedure call step (rather than a dynamic sub-procedure call step or graft step), calling the getStartIndex method on the vSubProcCase object that represents that sub-procedure will return -1.

Public Steps

When a step is defined with TIBCO Business Studio, the step can be designated a “public step.” Public steps provide the ability to specify that those steps can be used as "start case at" or "trigger event on" steps. This facility allows an application to limit case starting and event triggering to only those steps that have been designated as valid steps for those functions *if it wishes to do so*. TIBCO iProcess Server Objects does NOT enforce this limitation — it is the responsibility of the application to enforce this limitation if it so desires.

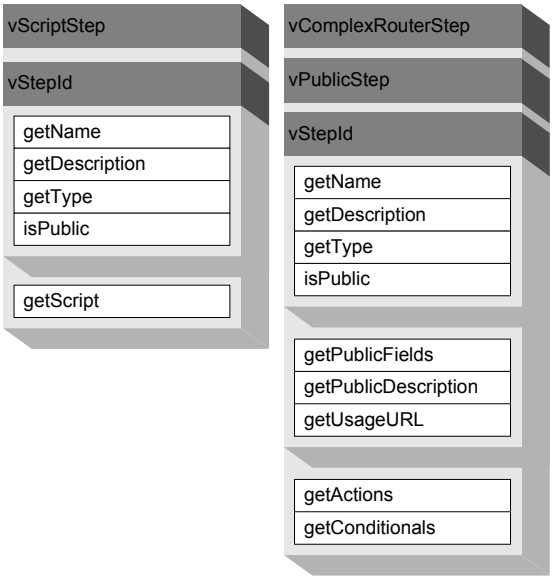
Note - Public steps are available only if you are using a TIBCO iProcess Engine.

The following table lists the types of steps that can be designated public steps when they are defined with TIBCO Business Studio:

Can be Public Step *(derive from vPublicStep)*	**Cannot be Public Step** *(derive from vStepId)*
vNormalStep	vEISStep
vComplexRouterStep	vScriptStep
vEventStep	vAutoStep[a]
vEAIStep	vOpenClientStep[a]
vSubProcCallStep	
vDynamicSubProcCallStep	
vGraftStep	

a. Not available if using a version i10 or newer TIBCO iProcess Objects Server.

Step types that cannot be public steps derive from the **vStepId** Value Object (see vScriptStep in the illustration). Step types that can be public steps derive from the **vPublicStep** Value Object (see vComplexRouterStep in the illustration).



The **vStepId** object has an **isPublic** method that returns True if the step has been defined as a public step in TIBCO Business Studio.

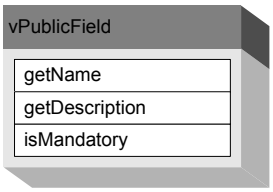
You can determine all of the public steps in a procedure by calling the **getPublicSteps** method on **sProcManager**. This method returns an array of vPublicStep objects, one for each step in the procedure that has been defined as a public step.

The **vPublicStep** object provides access to information that was defined when the step was designated as a public step in TIBCO Business Studio:

- **getPublicFields** - This method returns an array of **vPublicField** objects, one for each field that has been designated as a public field on that public step. See the “Public Fields” section below for more information.

- **getPublicDescription** - Description of the public step (which may differ from the vStepId description). This is entered in TIBCO Business Studio when the step is designated a public step.
- **getUsageURL** - A URL that may be used as a link for additional information about the public step. (This may hold a string that can be used for any purpose desired by the procedure designer.)

Public Fields



For a step that has been designated as a public step, you can also specify fields as being "public fields". For each field on a public step that has been designated as a public field, an **vPublicField** object is created.

Public Fields are provided so that an application can identify mandatory and optional input fields (based on the vPublicField.isMandatory flag). Again, it is up to the application to enforce whether data input into a public field is mandatory. TIBCO iProcess Server Objects does NOT enforce this requirement, nor return errors if data is not entered into mandatory fields.

The **vPublicField** object contains the following methods:

- **getName** - Returns the name given the public field.
- **getDescription** - Returns a description of the public field.
- **isMandatory** - Flag indicating whether or not this public field is mandatory. As described above, data entry in public fields that are flagged as mandatory is not enforced by the TIBCO iProcess Server Objects. It is up to the application to enforce this requirement.

Retrieving Public Field Objects

The vPublicField object is a dependent object on the vPublicStep object. Therefore, to retrieve vPublicField objects from the server when step objects are retrieved (with either getSteps or getPublicSteps on sProcManager), you must pass True in the *aIsWithPublicFields* parameter on the **vStepContent** object when calling getSteps or getPublicSteps.

7

Case Management

Starting a Case

A case is defined as an instance of a procedure. Therefore, starting a case means to create an instance of a procedure. This is done with the **startCase** method (available on both the **sUser** and **sWorkQ** objects). This method takes the form:

```
String startCase(String aProcTag,
                 String aDescription,
                 SWSubProcPrecedenceType aSubProcPrecedence,
                 String aStartStepName,
                 boolean aReleaseItem,
                 boolean aValidateFields,
                 vField[] aFields)
```

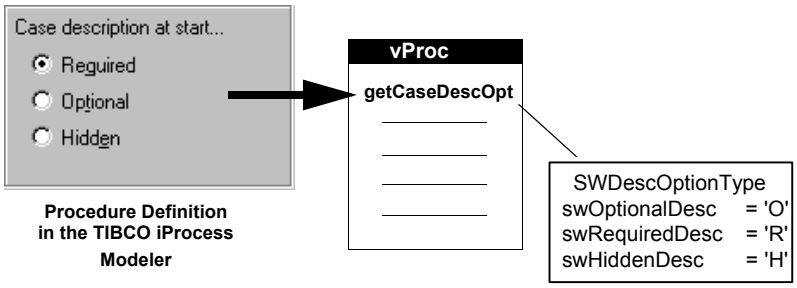
*Note - This is only one of the available signatures for **startCase** — see the on-line help system for all of the variations.*

By default, the case starts on the first step defined in the procedure — the name of this step is available with the **vProc.getStartStepName** method. However, you can optionally cause the procedure to start on a step other than the one specified in the procedure definition — this is done by providing the *aStartStepName* parameter when calling the **startCase** method.

*Note - You cannot directly start a case (with **startCase**) of a procedure that is defined as a sub-procedure (if it is a sub-procedure, the **isSubProc** method on **vAProc** will return True). Sub-procedures can only be started from a sub-procedure call step, dynamic sub-procedure call step, or graft step. See* *"Sub-Procedures" on page 52* *for more information.*

Case Description

The procedure definition (defined with TIBCO Business Studio) specifies whether or not a description must be specified when a case of the procedure is started.



This also determines if a description must be passed in the *aDescription* parameter when the **startCase** method is called, as follows:

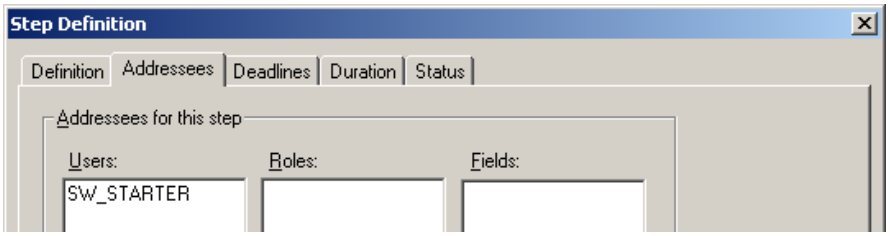
- If the procedure definition specifies that the description is **required** (vProc.getCaseDescOpt = swRequiredDesc), the *aDescription* parameter must contain a string.
- If the procedure definition specifies that the description is **optional** or **hidden** (vProc.getCaseDescOpt = swOptionalDesc or swHiddenDesc), the *aDescription* parameter can contain an empty string (“”).

Keeping/Releasing the Start Step

The **startCase** method provides an *aReleaseItem* parameter that allows you to specify that the start step be automatically released when the case is started. Note that this parameter is relevant *only* if the user starting the case is the addressee of the start step.

The addressee must be defined in one of the following ways on the Step Definition Addressee Tab in TIBCO Business Studio:

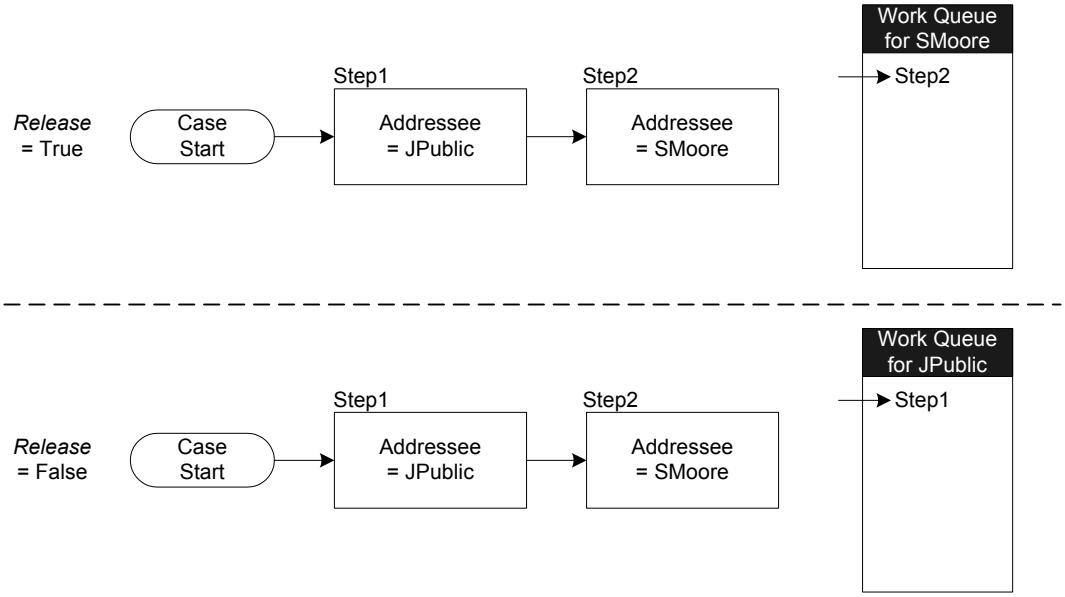
- Explicitly - The user’s name is entered in the Users column.
- SW_STARTER is listed in the Users column.
- A role name is listed in the Roles column, and the user starting the case is assigned to that role.



The *aReleaseItem* parameter is ignored if the user starting the case is not the addressee of the start step, or if the Fields column is used to specify the addressee of the start step.

- If *aReleaseItem* = **True** (the default), when the case is started, the start step is automatically released at the same time the case is started. This causes the case to automatically proceed to the second step, resulting in the work item for the second step appearing in the work queue of the addressee of the second step (see the illustration below).

- If *aReleaseItem* = **False**, the work item representing the start step is placed in the work queue of the addressee of the start step. This is always the behavior if the addressee is not the user starting the case.



The *aReleaseItem* flag is ignored if you specify a start step (with the *aStartStepName* parameter) other than the first step in the procedure.

Validating Markings on the Start Step (iProcess Modeler Forms Only)

When a form is created with the iProcess Modeler, the markings that are included on the form are given a "type" designation, indicating whether data in that field/marking is required, optional, calculated, etc. For example, the form might have a **First Name** field that is required, and a **Middle Name** field that is optional.

The type designation that is assigned to the field when it is added to the form in TIBCO Business Studio is available through the **vFMarking.getType** method for that particular marking on the form. The types for a marking are enumerated in **SWMarkingType**, as shown below.

```
SWMarkingType
swOptional      = 'O'
swRequired      = 'R'
swHidden        = 'H'
swDisplay       = 'D'
swCalculated    = 'C'
swEmbedded      = 'E'
```

The **startCase** method contains an *aValidateFields* parameter that allows you to specify whether or not to validate the fields/markings on the form in the start step, based on the marking types defined on the form:

- If *aValidateFields* = **True**, the case start will validate that the markings designated as **swRequired** on the form of the start step have values and are sent to the server (using the *aFields* parameter). It also verifies that no fields marked as **swDisplay** on the form are sent to the server. The marking's type designation can be obtained with the **vFMarking.getType** method. They are defined by the enumeration type **SWMarkingType**.
- If *aValidateFields* = **False** (the default), it bypasses the enforcement of marking types on the form.

This is probably most relevant when you are starting a case with field data (field values are passed in the *aFields* parameter) and the *aReleaseItem* flag is set to True (see the previous sections).

Note -The aValidateFields parameter is only applicable if you are using iProcess Modeler-produced forms (it validates markings, which are only applicable on iProcess Modeler forms).

Sub-Procedure Precedence

The **startCase** method provides an *aSubProcPrecedence* parameter that allows you to specify the order in which sub-procedure statuses are looked for when sub-procedures are launched from the main procedure. The sub-procedure precedence is enumerated in the **SWSubProcPrecedenceType** enumeration, as shown below:

```
SWSubProcPrecedenceType
  swPrecedenceR       = '0'
  swPrecedenceUR      = '1'
  swPrecedenceMR      = '2'
  swPrecedenceUMR     = '3'
  swPrecedenceMUR     = '4'
```

This enumeration allows you to specify that sub-procedure statuses be looked for in a specific order:

- swPrecedenceR - Released status only
- swPrecedenceUR - Unreleased > Released
- swPrecedenceMR - Model > Released
- swPrecedenceUMR - Unreleased > Model > Released
- swPrecedenceMUR - Model > Unreleased > Released

For example, if swPrecedenceUR is passed in the *aSubProcPrecedence* parameter, the engine will look for an unreleased status of the sub-procedure to start. If there isn't an unreleased status, it will look for a released status.

The default is to only look for released sub-procedures. Therefore, if a startCase method signature that does not include the *aSubProcPrecedence* parameter is used, only sub-procedures with a released status will be started.

If the specified (or default) statuses of a sub-procedure cannot be found, the error message "Sub-case started of a procedure that isn't a sub-procedure" is written to the **sw_warn** file.

Why isn't the Started Case Appearing in the Work Queue?

After starting a case, the work item representing the case you just started may not immediately appear in the work queue. This is because the work item is processed by the background process in the TIBCO iProcess Engine. Until the TIBCO iProcess Engine has completed its processing, the work item will not appear in the work queue.

Obtaining the Case Number of a Case that was just Started

When a case is started with **startCase**, it is assigned a "case number" that can be used for purposes such as tracking, filtering, sorting, etc. This number is available in the following ways:

- It is returned by the **startCase** method
- In the **vCaseId.getCaseNumber** method
- In the **SW_CASENUM** system field

The availability of the case number depends on which engine you are using:

- **TIBCO iProcess Engine** - With this engine, the case number is available immediately after the case is started.
- **TIBCO Process Engine** - With this engine, the case number is *not* available immediately. The work item that appears in the user's work queue will initially show a case number of 0 (zero). It will remain 0 for an indeterminate period of time. The case number is generated by the background process, so the amount of time it takes to generate it is determined by how frequently the background process "wakes up" and processes instructions from the TIBCO iProcess Objects Server. After the background process generates the case number, it is then available with the **getCaseNumber** method.

  If you are in a situation where you need the case number before the background process can provide it, the following can be used as a work around: A "case number synchronization" step could be added to the procedure definition just after the procedure start step. The addressee for the "case number synchronization" step could be a user such as "CaseAdmin". When the start case has been processed by the background, a work item will appear on the CaseAdmin's work queue. Application code could then be written to get (and lock) the work item from CaseAdmin's work queue to get the case number (**vCaseId.getCaseNumber**) and do whatever processing is necessary, then release the work item so it will go to the next step.

  Another alternative is to add a custom field that has a unique identifier provided by the user or some external system. Then display or search on this number. After the case start has completed and the work item has reached a queue, then you can associate the case number (TIBCO unique) to the customer's unique number.

  See "TIBCO Process / iProcess Engine" on page 4 for more information about the two types of engines.

Determining Who Can Start a Case

By default, when a procedure is created on a node, *all* users on the node have the authority to start cases against that procedure (assuming the procedure is "released").

You can specify the users who can start a case of a procedure by using TIBCO Business Studio.

If a user name or role is specified using this function in TIBCO Business Studio, case-start access to this procedure is limited to only the users, groups, or roles specified, or to the users for which the expression(s) evaluate to True. If no one is designated as having access through this function, it defaults to giving *everyone* access.

*Note - If a procedure is "unreleased" (**vProc.getStatus = "swUnreleased"**), it can only be started by the procedure owner or the IPEADMIN user. It must have a status of "**swReleased**" for other users to be able to start cases of that procedure. If the procedure's status is "**swIncomplete**" or "**swWithdrawn**", no one, not even the IPEADMIN user, can start cases of that procedure. (For information about the IPEADMIN user, see* *"User Authority" on page 190.**)*

You can determine which users, groups, or roles have permission to start a procedure by calling the **getStartByUserRef** method on **vProcDef**. (This cannot be set through TIBCO iProcess Server Objects; it can only be set through TIBCO Business Studio.)

The **getStartByUserRef** method returns a **vAccessUserRef** object, which contains the following methods:

- **getUserNames** - Returns an array of the users or groups who have authority to start a case of the procedure.
- **getRoleNames** - Returns an array of the roles that have authority to start a case of the procedure.
- **getExpressions** - Returns an array of expressions that indicate the attribute values the user must have to be able to start the case. In the example above, if the user's DEPARTMENT attribute is "legal" or "hr", that user has authority to start a case of the procedure.

If all three of these arrays are empty, *all* users have authority to start a case of the procedure.

Which Procedures can a User Start?

You can also determine the procedures for which a user can start a case by calling the **getStartProcIds** method on the **sUser** object. This returns an array of **vProcId** objects, one for each procedure the user is authorized to start. From the **vProcId** object, you can call **getTag** to obtain the tag for the procedure, then use that as an input parameter with the **startCase** method.

Obtaining Lists of Cases

As cases of a procedure are started and finished, a list of these cases (both active and completed) is maintained by the TIBCO iProcess Engine. The **sCaseManager** Server Object contains a number of methods that allow you to access these cases in different ways. They are:

- **getACases** - This returns an array of **vACase** objects. This method is typically used if you are interested in only one or a few of the cases for a particular procedure, as you must provide the case numbers for the cases you want returned.
- **getACaseList** - This returns a pageable list of **vACase** objects. This method is used to obtain a list for all cases of a procedure. It provides parameters that allow you to filter and/or sort the list of cases returned, as well as specify that the list be "held" so that it can be retrieved at a later time with the **getACaseListHeld** method (see below).

  *Note - Lists of cases obtained with the **getACaseList** method always includes cases from all versions of the procedure. There currently is no means of filtering the list to include only cases from a specific version of the procedure.*
- **getACaseListHeld** - This returns a pageable list of **vACase** objects that you had previously obtained and held with the **getACaseList** method (see above). It requires you to provide a "held ID" to identify the pageable list of cases.

The **getACases** and **getACaseList** methods both require that you pass a procedure "tag" in the *aProcTag* parameter. This identifies the particular procedure from which you want to obtain cases. The procedure tag is available by calling the **getTag** method from the **vProcId** object (the **makeTag** method can also be used to construct a tag if you know all of the components that make up the tag). The **vProcId** object can be obtained in the following ways:

- **vCase.getProcId** - Identifies the procedure of which the case is an instance.
- **sUser.getStartProcIds** - Identifies the procedures the user can start.
- **sUser.getAuditProcIds** - Identifies the procedures the user can audit.
- **sProcManager.getProcIds** - Returns **vProcId** objects for either all procedures on the node, or for specific procedures.
- **sCaseManager.getProcIds** - Returns **vProcId** objects for all procedures on the node.

Determining the Number of Cases in a Procedure

There are a number of methods available that tell you how many cases there are of a procedure. These methods are available on the **vProcSummary** object, which you can get by calling **vAProc.getProc-Summary**. The methods available are:

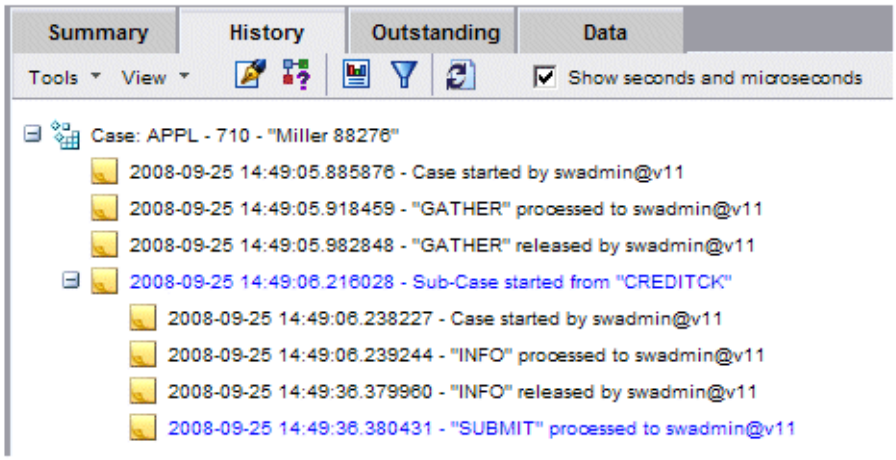
- **vProcSummary.getCaseCnt** - The total number of cases in the procedure, both active and closed.
- **vProcSummary.getActiveCnt** - The number of active cases (**vACase.isActive=True**) in the procedure.
- **vProcSummary.getClosedCnt** - The number of closed cases (**vACase.isActive=False**) in the procedure.

You can also determine the number of cases in a procedure that will satisfy a specific filter expression. This can be used to determine the number of cases before calling a method that would return the cases in a pageable list. This can be done using the following method:

- **sCaseManager.getCaseCnt** - Returns the total number of cases that satisfy the specified filter expression.

Auditing Cases

The system maintains an “audit trail” that provides information about a case’s progress through a procedure, that is, which steps in the procedure have been processed and who processed them. An example of how this information might be presented to the user is shown below (this example is from the TIBCO iProcess Workspace (Browser)):



The entries that are shown in blue are the steps that are currently outstanding.

There are two types of audit trail entries:

- **System-defined** - These are added to the audit trail by the system each time an action of some sort is performed on the step in the case. These messages are pre-defined in *SWDIR***etc*****language**.**lng****audit.mes** (Windows) or $SWDIR/**etc**/***language***.**lng**/**audit.mes** (UNIX). An excerpt from the **audit.mes** file is shown below:

```
000:Case started by %USER
001:"%DESC" processed to %USER
002:"%DESC" released by %USER
003:Deadline for "%DESC" expired for %USER
  :
  :
```

  The three-digit number on the left is the *MsgId* of the audit trail message. The system reserves *MsgIds* 000-255 for system use.

- **User-defined** - These are added to the audit trail of a live case when you invoke the **addCaseAuditEntry** method, which is available from the **sUser** and **sWorkQ** objects. These messages must be predefined in *SWDIR***etc*****language**.**lng****auditusr.mes** (Windows) or $SWDIR/**etc**/***language***.**lng**/**auditusr.mes** (UNIX). For information about adding user-defined audit entries, see “Adding User-defined Audit Trail Entries” on page 80.

Determining who can Audit Cases of a Procedure

By default, when a procedure is created on a node, *all* users on the node have the authority to audit cases of that procedure.

You can specify the users who can audit cases of the procedure by using TIBCO Business Studio.

You can specify who has Case Administration authority. If a user, group, or role name is specified, the ability to audit this procedure is limited to *only* the users, groups, or roles specified, or the users for which the expression(s) evaluates to True. Note that having system administrator authority (MENUNAME = ADMIN) does NOT automatically give you access to audit trail data — if users are given access through this dialog, users with a MENUNAME of ADMIN must be explicitly listed to have access. (The IPEADMIN user always has access to the audit trail of any procedure. (For information about the IPEADMIN user, see "User Authority" on page 190.) If no one is designated as having access through this function, it defaults to giving *everyone* access.

Using the TIBCO iProcess Server Objects, you can determine which users, groups, or roles have permission to audit cases of a procedure by calling the **getAdminByUserRef** method on **vProcDef**. (This cannot be set through TIBCO iProcess Server Objects; it can only be set through TIBCO Business Studio.)

The **getAdminByUserRef** method returns a **vAccessUserRef** object, which contains the following methods:

- **getUserNames** - Returns an array of the users or groups who have authority to audit cases of the procedure.
- **getRoleNames** - Returns an array of the roles that have authority to audit cases of the procedure.
- **getExpressions** - Returns an array of expressions that indicate the attribute values the user must have to be able to audit cases. In the example above, if the user's DEPARTMENT attribute is "legal" or "hr", that user has authority to audit cases of the procedure.

If all three of these arrays are empty, *all* users have authority to audit cases of the procedure.

Which Procedures can a User Audit?

You can determine which procedures a user can audit by calling the **getAuditProcIds** method on **sUser**. This method returns an array of **vProcId** objects, one for each procedure the user has permission to audit.

Audit Step Objects

Information about the progress of processing steps in a case (considered "audit data" or the "audit trail") is maintained on the TIBCO iProcess Engine. This information is available in TIBCO iProcess Server Objects in the form of **vAuditStep** objects.

As a case is being processed, one “audit step” is generated each time an action is performed in that case.

The audit step *action* (available with the **getAction** method) indicates what type of action occurred. These are enumerated by the **SWAuditActionType** object.

The first action is always the “start case” action (**swStartCase**). The **swProcessedTo** action indicates that the step has been “processed to” the addressee of the step. Typically, this means that a work item for that step has been sent to the work queue of the addressee.

The **getMessage** method on **vAuditStep** returns the actual message that appears in the audit trail. This message will have any **%USER** and **%DESC** variables resolved that are part of the message in the **audit.mes** or **auditusr.mes** files (see page 69 for information about these files). For example, if the message in audit.mes is:

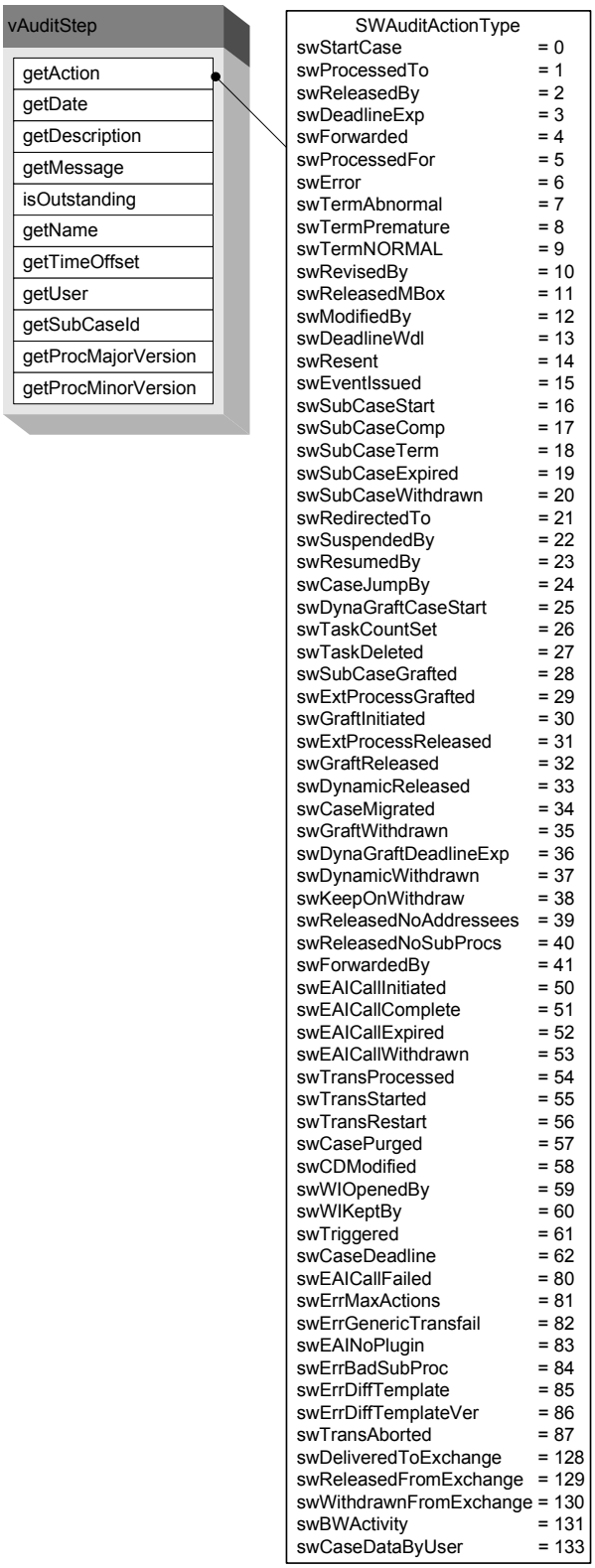
%DESC processed to %USER

the string returned by getMessage would be (assuming a step description of "Final Approval" and a user name of susieq):

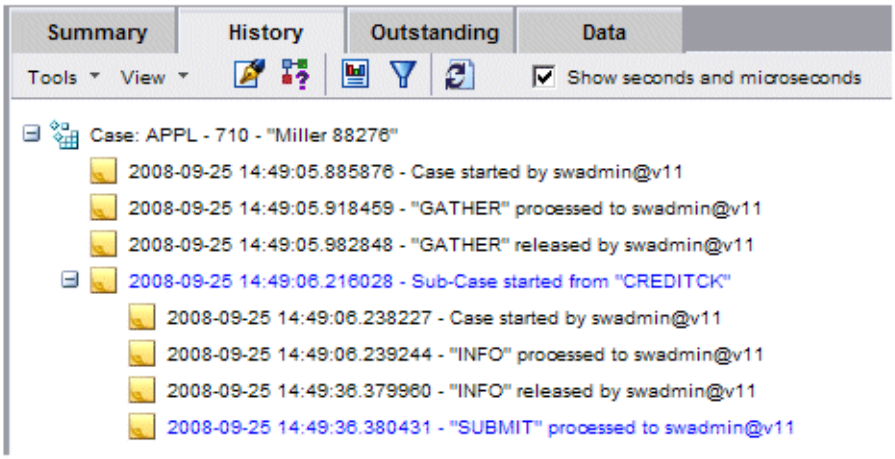
"Final Approval" processed to susieq

The %DESC and %USER variables are replaced with the step description and the name of the user who performed the action, respectively.

The **isOutstanding** method on **vAuditStep** allows you to determine if the step is *outstanding*, meaning the step has been “processed to,” but it has not been released.



The audit information in the **vAuditStep** objects is used to present the "audit trail" to the user. For example:



The entries that are shown in blue are the steps that are currently outstanding.

Some of the entries in the audit trail may represent steps in a sub-procedure. For information about auditing sub-procedures, see "Auditing Sub-Procedures" on page 74.

Getting Audit Step Objects

You can get audit step objects in one of two ways:

- **sCaseManager.getAuditSteps** - This causes a message to be sent to the TIBCO iProcess Objects Server, requesting an array of **vAuditStep** objects for the specified case.
- **vACase.getAuditSteps** - This returns an array of **vAuditStep** objects from the local **vACase** object. As the **vAuditStep** objects are dependent objects on **vACase** (see the illustration below), you must use a "content" object, **vACaseContent**, to request that the **vAuditStep** objects also be retrieved from the server when the **vACase** object is retrieved.

  The **vACaseContent** content request object constructor has an *aIsWithAuditData* parameter that is used to specify whether or not audit data (**vAuditStep** objects) are also returned when **vACase** objects are retrieved:

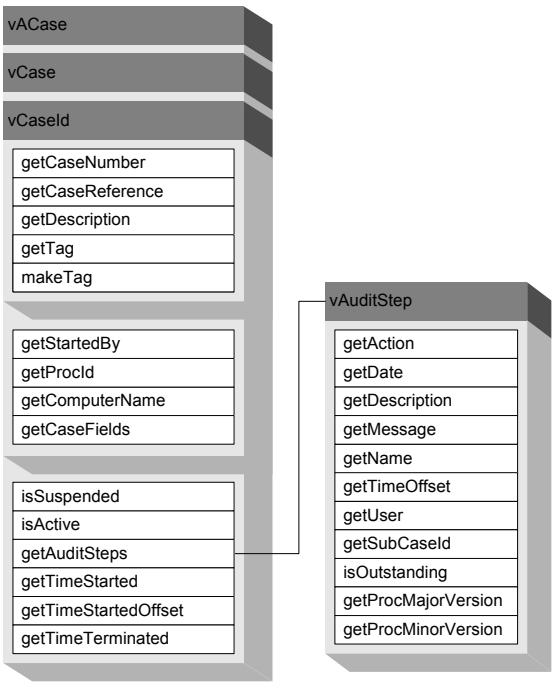
```
vACaseContent(boolean aIsWithAuditData,
              boolean aIsAuditAscending,
              String aAuditFilterExpression)
```

Passing True in the *aIsWithAuditData* parameter requests that audit data be returned.

You can also specify, using the *aIsAuditAscending* parameter on the **vACaseContent** object, that the audit data be listed in ascending order (the **swStartCase** action is listed first) or descending order (the last action processed is listed first).

See "Retrieving Dependent Objects" on page 160 for more information about using content request objects.

Once you have retrieved **vACase** objects with the dependent **vAuditStep** objects, calling the **vACase.getAuditSteps** method returns the audit steps requested for that case.



Configuring Audit Trail Strings

Part of the text that appears in the audit trail message is obtained from the step description and the name of the user who performed the action. However, some actions don't have corresponding steps from which a description can be obtained (e.g., case suspension). Also, some actions are performed by the system (e.g., case termination/closure); these actions do not have a corresponding user name that can be written to the audit trail message. Because of this, TIBCO iProcess Objects Server configuration parameters are provided that contain default values that are written to the **Description**, **Name**, and **User** properties on **vAuditStep** for these types of actions. You can change these default values to fit your particular needs.

The table below lists the audit actions that have configuration parameters:

Action	Configuration Parameter	Written to this vAuditStep Property	Default
swStartCase	StartCaseDescription	Description	"Case Start"
	StartCaseStepName	Name	"Case Start"
swTermNORMAL	TerminationDescription	Description	"Termination"
	TerminationStepName	Name	"Termination"
	TerminationUser	User	"System"
swTermAbnormal	TerminationDescription	Description	"Termination"
	TerminationStepName	Name	"Termination"
	TerminationUser	User	"System"
swTermPremature	TerminationDescription	Description	"Termination"
	TerminationStepName	Name	"Termination"

Action	Configuration Parameter	Written to this vAuditStep Property	Default
swSuspendedBy	SuspendedDescription	Description	“Case Suspended”
	SuspendedStepName	Name	“Case Suspended”
swResumedBy	ResumedDescription	Description	“Case Activated”
	ResumedStepName	Name	“Case Activated”
swCaseJumpBy	JumpToStepName	Name	“Jump To”

Note that the configuration parameters that pertain to case suspension, resume, and jump to, are applicable only if you are using a TIBCO iProcess Engine. For more information about using configuration parameters, see the *TIBCO iProcess Objects Server Administrator’s Guide.*

Auditing Sub-Procedures

There are a number of audit action types (**SWAuditActionType**) that are specific to sub-procedures. They are:

SWAuditActionType	Value	Description
swSubCaseStart	16	Sub-case has been started.
swSubCaseComp	17	Sub-case has completed.
swSubCaseTerm	18	Sub-case has terminated prematurely.
swSubCaseExpired	19	Sub-case deadline has expired.
swSubCaseWithdrawn	20	Sub-case has been withdrawn.
swDynamicReleased	33	Dynamic sub-procedure call step has been released.
swDynamicWithdrawn	37	Dynamic sub-procedure call step has been withdrawn.
swErrBadSubProc	84	Error - Invalid sub-procedure.
swErrDiffTemplate	85	Error - Different templates.
swErrDiffTemplateVer	86	Error - Different template versions.

Grouping Sub-Cases in the Audit Trail

The **getSubCaseId** method returns a system-generated string that can be used to group audit steps by case when steps from both main procedures and sub-procedures are being processed simultaneously. This method is available on both the **vAuditStep** and **vSubProcCase** objects. By comparing the string returned by the getSubCaseId method from both of these objects, you can group together the audit steps from a particular sub-case.

If the audit step is in the main procedure, the **getSubCaseId** method returns an empty String.

Filtering Audit Data

Filtering audit data allows you to minimize the amount of audit data that is retrieved from the server, making your client application more efficient. This is accomplished by creating a filter criteria expression, then passing this string in the *aAuditFilterExpression* parameter when constructing the **vACaseContent** object.

If the *aIsWithAuditData* parameter is set to True, only **vAuditStep** objects that satisfy the filter expression are returned from the server with the **vACase** object.

Creating an Audit Filter Expression

When creating an audit filter expression, you can AND together the following criteria to create a filter expression:

- **AUDIT_TYPE** - A list of audit-entry types to return or not return, depending on the operator used (= or !=).
- **USER_NAME** - A list of user names (or sub-case IDs).
- **STEP_NAME** - A list of step names.
- **STEP_DESC** - A list of step descriptions.
- **DATE_RANGE** - A range of dates.
- **FILTER_FLAGS** - Special condition flags.

Details about these criteria are provided below.

- **AUDIT_TYPE** - A list of audit entry types to return or not return, depending on the operator used (= or !=).

  AUDIT_TYPE=[type1|type2|type3|....|typen]

  --or--

  AUDIT_TYPE!=[type1|type2|type3|....|typen]

  where *type* identifies the type of each audit trail entry you want to appear in the audit trail. Note that the iProcess Objects Server provides equates for some of the audit types, but not all of them. Each audit type also has a numeric value defined that can be used in **AUDIT_TYPE**. When specifyng the type(s) in **AUDIT_TYPE**, you can use either the equates, the numeric values, or a combination of both.

  The following table lists all available audit type numeric values, and equates where available.

Numeric Value	Audit Trail Message	Description	Equates
0	Case started by %USER	Case started	AT_START
1	"%DESC" processed to %USER	Work item sent to queue	AT_SENT
2	"%DESC" released by %USER	Work item released	AT_RELEASED
3	Deadline for "%DESC" expired for %USER	Work item deadline expired	AT_EXPIRED
4	"%DESC" forwarded to %USER	Work item forwarded	AT_FORWARD
5	"%DESC" processed on behalf of %USER	Autostep processed	AT_AUTOSTEP

Numeric Value	Audit Trail Message	Description	Equates
6	Error - "%DESC" not found	Error - step not found	AT_NOTFOUND
7	Case terminated abnormally	Case terminated abnormally	AT_BADTERM
8	Case terminated prematurely by %USER	Case terminated prematurely	AT_CLOSED
9	Case terminated normally	Case terminated normally	AT_NORMTERM
10	"%DESC" revised by %USER	Case data revised	AT_REVISED
11	"%DESC" released from queue by %USER	Work item released directly from queue	AT_QRELEASED
12	n/a	Work item modified	n/a
13	"%DESC" withdrawn from %USER	Work item withdrawn from queue	n/a
14	"%DESC" resent to %USER	Work item resent to queue	AT_RESENT
15	"%DESC" event issued by %USER	Event triggered	AT_EVENT
16	Sub-Case started from "%DESC"	Sub-case started	AT_SUBSTART
17	Sub-case started from "%DESC" completed	Sub-case terminated normally	AT_SUBEND
18	Sub-case started from "%DESC" terminated abnormally	Sub-case terminated abnormally	AT_SUBBADEND
19	Deadline for sub-case started from "%DESC" expired	Sub-case deadline expired	AT_SUBEXPIRED
20	Sub-case started from "%DESC" closed	Sub-case closed prematurely	AT_SUBCLOSED
21	"%DESC" redirected to %USER	Work item redirected	AT_REDIRECTED
22	Case Suspended by %USER	Case suspended	n/a
23	Case Resumed by %USER	Case activated/resumed	n/a
24	"%DESC" Case Jump by %USER	Case jumped	n/a
25	"%DESC" Sub-Case started (using array element %STEPNAME)	Sub-case started using array element	n/a
26	Task count '%STEPNAME' received for "%DESC"	Task count received	n/a
27	Task count decremented for "%DESC"	Task count decremented	n/a
28	Sub-Case grafted to "%DESC"	Sub-case grafted	n/a
29	External process "%USER" grafted to "%DESC"	External process grafted	n/a
30	"%DESC" initiated	Graft step initiated	n/a
31	External process "%USER" released	Grafted external process released	n/a
32	"%DESC" released, all tasks complete	Graft step released - all tasks complete	n/a

Numeric Value	Audit Trail Message	Description	Equates
33	"%DESC" released, all sub-cases complete	All cases started from multi sub-procedure call step complete	n/a
34	Case migrated from Procedure v%STEPNAME to v%DESC by "%USER"	Case migrated to new procedure version	n/a
35	Sub-cases, grafted to "%DESC", closed	Grafted sub-cases closed	n/a
36	Deadline for "%DESC" expired	Graft step or multi sub-procedure step deadline expired	n/a
37	Sub-cases, started from "%DESC", closed	Graft step or multi sub-procedure step closed or withdrawn	n/a
38	"%DESC" withdrawn, outstanding items not deleted	Graft step or multi sub-procedure step withdrawn, outstanding items not deleted	n/a
39	No addressees defined for step "%DESC" - automatically released	No addressees defined for step - automatically released	n/a
40	No sub-procedures defined for step "%DESC" - automatically released	No sub-procedures defined for step - automatically released	n/a
50	"%DESC" EAI call-out initiated ("%USER")	EAI step initiated	n/a
51	"%DESC" EAI call-out completed ("%USER")	EAI step completed	n/a
52	Deadline for EAI Step "%DESC" expired	EAI step deadline expired	n/a
53	EAI Step "%DESC" withdrawn	EAI step withdrawn	n/a
54	Commit Point "%DESC" reached	Transaction commit step reached	n/a
55	New Transaction started from "%DESC"	Transaction commit step started	n/a
56	New Transaction start retried from "%DESC"	Transaction commit step retried	n/a
57	Case Purged	Case deleted from system	n/a
58	"%DESC" Case data modified by %USER	Case data modified	n/a
59	"%DESC" opened by %USER	Work item opened	n/a
60	"%DESC" kept by %USER	Work item kept	n/a
61	%DESC" Triggered: "%STEPNAME" event issued by %USER	Trigger event issued	n/a
62	%DESC" case deadline event issued by %USER	Case deadline event issued	n/a

Numeric Value	Audit Trail Message	Description	Equates
80	"%DESC" EAI call-out failed ("%USER")	EAI step failed	n/a
81	Workflow may have an infinite loop (at "%DESC") - reached max actions per transaction (%USER)	EAI step possible infinite loop - reached maximum number of actions	n/a
82	Error, workflow transaction aborted because of a system failure - check sw_warn/sw_error logs	Error - transaction aborted because of system failure	n/a
83	The run-time plug-in for EAI Type "%USER" (used by step "%DESC") not registered on all servers or failed to load/initialise correctly.	EAI step runtime plugin failed to load	n/a
84	Invalid sub-procedure "%USER" specified for "%DESC" - check sw_warn/sw_error logs	Invalid sub-procedure	n/a
85	"%DESC" and sub-procedure "%USER" are not based on the same parameter template - check sw_warn/sw_error logs	Parameter template name mismatch	n/a
86	"%DESC" and sub-procedure "%USER" are not based on the same version of parameter template - check sw_warn/sw_error logs	Parameter template version mismatch	n/a
87	Transaction Aborted at "%DESC"	Transaction commit step aborted	n/a
128	%DESC" delivered to Exchange recipient %USER	Delivered to Exchange recipient	n/a
129	%DESC" release received from Exchange recipient %USER	Release received from Exchange recipient	n/a
130	%DESC" withdrawn from Exchange recipient %USER	Withdrawn from Exchange recipient	n/a
131	BusinessWorks Activity Audit "%DESC" processed by "%USER"	BusinessWorks activity audit	n/a
133	Case data changed by %USER	Case data changed by user	n/a

Note that *type* in **AUDIT_TYPE** can also be a numeric value that identifies a *user-defined audit trail message* (see "Adding User-defined Audit Trail Entries" on page 80 for information about how user-defined messages are defined).

or, *type* can be one of the following:

AT_SWTYPES - Include only TIBCO-defined audit entries.

AT_APPTYPES - Include only application-defined (custom) audit entries.

- **USER_NAME** - A list of up to five user names or sub-case IDs for which audit entries are to be returned. Note that wildcard characters * and ? can be used. A * indicates zero or more characters of any value at the position it appears. A ? indicates one character of any value at the posi-

tion it appears. Character matching is also case insensitive. Note that sub-case IDs can also be used with this criteria because the user name for a sub-procedure call step is a sub-case Id (see **getSubCaseId**).

USER_NAME=[username1|username2|....|username5]

--or--

USER_NAME=[sub-caseid1|sub-caseid2|....|sub-caseid5]

where:

username is a valid TIBCO user name, in the format <UserName>@<NodeName>. The user name can be specified with or without quotes.

sub-caseid is a valid sub-case ID.

- **STEP_NAME** - A list of up to five step names for which audit entries are to be returned. Note that wildcard characters * and ? can be used. A * indicates zero or more characters of any value at the position it appears. A ? indicates one character of any value at the position it appears. Character matching is also case insensitive.

  STEP_NAME=[stepname1|stepname2|....|stepname5]

  where:

  stepname is a valid step name. The step name can be specified with or without quotes.

- **STEP_DESC** - A list of up to five step descriptions for which audit entries are to be returned. Note that wildcard characters * and ? can be used. A * indicates zero or more characters of any value at the position it appears. A ? indicates one character of any value at the position it appears. Character matching is also case insensitive.

  STEP_DESC=[stepdesc1|stepdesc2|....|stepdesc5]

  where:

  stepdesc is a valid step description. The step description can be specified with or without quotes.

- **DATE_RANGE** - The DateTime value specifying a "from" and "to" range for audit entry dates. The range is inclusive of the "from" and "to" dates. If the from_DateTime parameter is omitted, the beginning of time is assumed. If the to_DateTime parameter is omitted, the current date and time is assumed. If either parameter is omitted, the hyphen must still be entered.

  DATE_RANGE=[from_DateTime - to_DateTime]

  where:

  from_DateTime = DateTime specifying the beginning of the date range. This is specified as a string. For example:

  "17/08/2000 01:00"

  to_DateTime = DateTime specifying the end of the date range. This is specified as a string. For example:

  "20/08/2000 05:00"

*Note - The order of the month and day in the date is specified in the **staffpms** file (see "Date Format" on page 178).*

- **FILTER_FLAGS** - Flags that specify special conditions that are outside of the usual realm of the filter criteria above.

  FILTER_FLAGS=[flag1|flag2]

  where:

  flag = AF_OUTSTANDING_ONLY and/or AF_ALL_SUBSTART

  AF_OUTSTANDING_ONLY = Include only "processed to" (or equivalent) entries for steps that are currently outstanding.

  AF_ALL_SUBSTART = Always return sub-case started entries regardless of whether or not they match other filter criteria.

Example Audit Filter Expression

("AUDIT_TYPE!=[AT_START|AT_SENT] AND STEP_DESC=[\"new case\"] AND DATE_RANGE=[- \"17/08/2000 01:30\"]")

Note - The filter criteria names (AUDIT_TYPE, USER_NAME, etc.) are case insensitive if your TIBCO iProcess Objects Server has CR 16694 implemented; if your TIBCO iProcess Objects Server does not include CR 16694, the filter criteria names must be all uppercase. Also, the values following each criteria must be enclosed in square brackets [ ].

Adding User-defined Audit Trail Entries

TIBCO iProcess Server Objects provides the ability to add user-defined audit trail entries to a live case by invoking the **addCaseAuditEntry** method, which is available on both the **sUser** and **sWorkQ** objects.

User-defined audit trail messages must be predefined in *SWDIR***etc*****language**.**lng****auditusr.mes** (Windows) or $SWDIR/**etc**/***language***.**lng**/**auditusr.mes** (UNIX). You must create (or add to) this file if you want to specify user-defined audit trail messages with the **addCaseAuditEntry** method.

User-defined audit trail messages must be in the format:

msg_num:msg_description

where:

msg_num is a decimal number in the range 256-999 that identifies the message. This number is used in the *MsgId* parameter with the **addCaseAuditEntry** method.

msg_description is a string that describes the event. It can contain the strings **%USER** and **%DESC**, which are replaced by the *UserName* and *StepDesc* strings, respectively, that are supplied with the **addCaseAuditEntry** method.

Below is an example of a user-defined audit trail message in the **auditusr.mes** file:

256:"%DESC" being worked on by %USER

Once the user-defined message is added to the **auditusr.mes** file, it can be added to the audit trail using the **addCaseAuditEntry** method. (Note - If you make a change to the **auditusr.mes** file, you must restart the iProcess Objects Server before the change will be recognized.)

For syntax details for **addCaseAuditEntry**, see the on-line help system.

Be aware that entries you add via the **addCaseAuditEntry** method may seemingly appear out of order when compared to system-added entries. This is because system-added entries are added by the background process, possibly causing a delay in their entry.

Custom Audit Trail Message Templates

You can obtain information about custom audit trail messages that have been added to the **auditusr.mes** file. The **getCustomAuditMsgDefs** method (available on **sUser** and **sWorkQ**) returns an array of **vAuditMsgDef** objects, one for each message that has been defined in the **auditusr.mes** file.

The **vAuditMsgDef** object contains the following methods:

- **getMsgNumber** - This returns the identifying number of the custom audit trail message represented by the **vAuditMsgDef** object.
- **getMsgTemplate** - This returns the "template" (i.e., the actual message) for the custom audit trail message represented by the **vAuditMsgDef** object.

Withdraw Outstanding Items / Jump To New Steps

You can specify that one or more outstanding items in a case family (a main case and all of its sub-cases) be "withdrawn" and that the process "jump to" one or more other steps in the case family, making those steps outstanding items. (To perform this functionality, you must be using a TIBCO iProcess Engine.)

Note - Outstanding items represent the steps at which the process flow is currently sitting. Normal steps (swNormal) that are outstanding result in a work item appearing in one or more work queues. All other step types (event steps, sub-procedure call steps, etc.), result in some other action, such as an external program being triggered, a sub-procedure being started, etc. These step types do not result in a work item appearing in a work queue, although they are still considered outstanding because the process flow is halted along that path of the process flow until whatever action was started by that step is complete.

A withdrawal / jump-to operation is performed with the **jumpTo** method on the **sCaseManager** object. The **jumpTo** method allows you to:

- Specify the set of outstanding items to withdraw. These items may be in the main case or any sub-case. See the "Determining Outstanding Items" section below for information about determining the currently outstanding items in the case. (You can also specify a '*' wildcard in the withdrawal list to withdraw all outstanding steps in the case family.)
- Specify the set of steps to "jump to", making these steps the new outstanding items. These steps may be in the main case or any sub-case. You can also optionally override the new outstanding item's default addressee, specifying one or more new addressees.

- Update case data when the withdrawal / jump to is performed. You can also optionally specify that work item data be updated in all outstanding items. (See "Case Data vs. Work Item Data" on page 166 for information about the difference between these types of data.)

  Note that if you are updating case data when performing a withdrawal / jump-to operation, the fields you want to update may reside in a sub-case. For details about identifying the fields in the sub-case, see the jumpTo method in the on-line help system.

You can withdraw / jump to the following types of steps:

- Normal (swNormal)
- Event (swEvent)
- EAI (swEAI)
- Sub-procedure (swSubProcCall)
- Dynamic sub-procedure (swDynamicSubProcCall)

You cannot withdraw an outstanding item representing a transaction control step, although you can jump to a transaction control step.

You cannot withdraw an outstanding item representing a graft step, nor jump to a graft step.

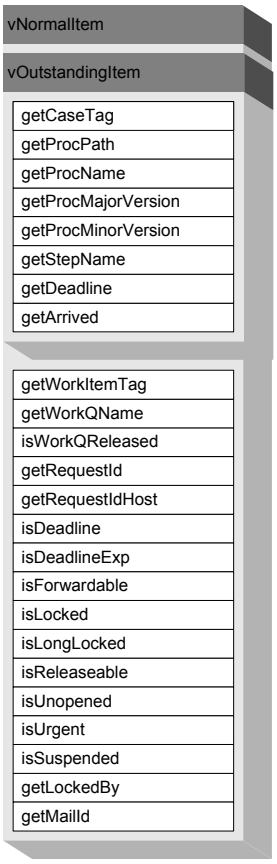
Determining Outstanding Items

The TIBCO iProcess Server Objects object model provides an "Item" class corresponding to each of the step types that can result in an outstanding item. Obtaining a list of the current outstanding items in a case family allows you to determine the items available for withdrawal with the **jumpTo** method.

The outstanding item objects are:

- **vNormalItem** - Normal step
- **vEventItem** - Event step
- **vEAIItem** - EAI step
- **vSubProcCallItem** - Sub-procedure call step
- **vDynamicSubProcItem** - Dynamic sub-procedure call step
- **vGraftItem** - Graft step (Note that graft steps can result in outstanding items, although you cannot withdraw this type of outstanding item.)
- **vTransactionControlItem** - Transaction control step (Note that transaction control steps can result in outstanding items, although you cannot withdraw this type of outstanding item.)

All of the outstanding item objects listed above derive from the **vOutstandingItem** Value Object (an example — **vNormalItem** — is shown on the right).



In order to determine which work items are currently outstanding, for potential withdrawal, the **getOutstandingItems** method on **sCaseManager** is used. This method returns an array of **vOutstandingItem** objects, one for each step that is currently outstanding in the case family (the main case and all of its sub-cases).

The **getOutstandingItems** method provides a couple of parameters that allow you to limit the amount and type of outstanding items that are returned by the method. They are:

- *aOutstandingItemContent* - This parameter allows you to specify the types (vNormalItem, vEventItem, etc.) of outstanding item objects to return. This requires that you construct a **vOutstandingItemContent** object and pass it in the method call.
- *aIncludeSubProcs* - This Boolean parameter allows you to specify whether or not to include outstanding items that are in sub-procedures that have been started from the main case.

After obtaining the array of vOutstandingItem objects with the getOutstandingItems method, determine which of those outstanding items you want to withdraw. On the vOutstandingItem object for each item you want to withdraw, call the **getProcPath** method. This returns the path to that particular outstanding item, which can then be used in the "withdraw list" when calling the **jumpTo** method. See the next section for more information about the ProcPath

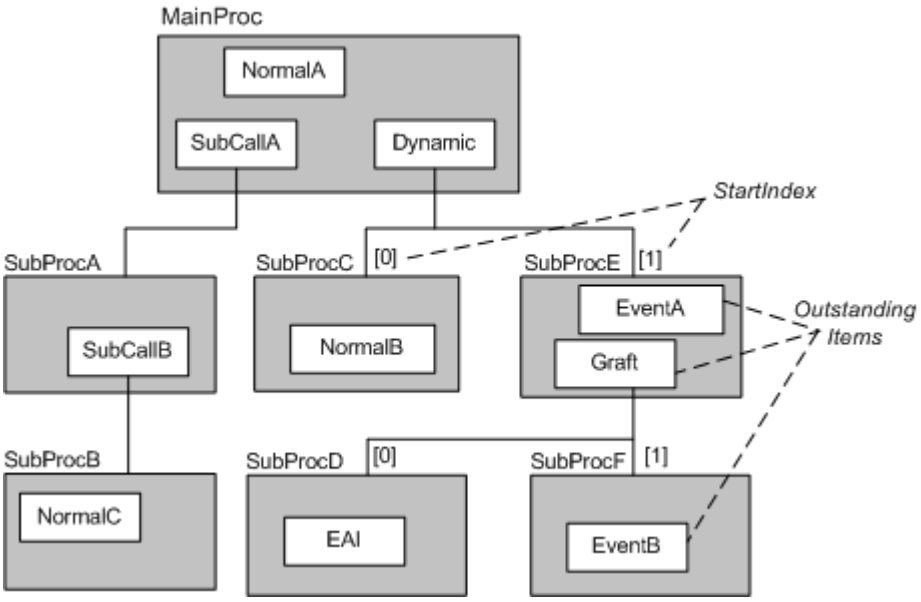
*Note - You can also include the ProcPath to an outstanding sub-procedure in the **jumpTo** withdraw list. This causes all outstanding items in that sub-procedure to be withdrawn. See* *"SubProcPath to a Sub-Case" on page 57* *for information about obtaining the ProcPath to a sub-procedure.*

ProcPath to Outstanding Items

Every outstanding item contains a "ProcPath" that provides a path from the main procedure to that specific outstanding item. This ProcPath can be used in the *WithdrawList* parameter with the **jumpTo** method to identify the outstanding items you want to withdraw.

The ProcPath for an outstanding item is obtained by calling the **getProcPath** method on the **vOutstandingItem** object that represents the outstanding item you would like to withdraw. This method returns a string that provides the path from the main procedure to the outstanding item.

The illustration below shows how the ProcPath is constructed for a variety of outstanding items that are several levels below the main procedure.



Step	ProcPath
NormalA	“NormalA”
NormalB	“Dynamic[0]|NormalB”
NormalC	“SubCallA|SubCallB|NormalC”
EventA	“Dynamic[1]|EventA”
EventB	“Dynamic[1]|Graft[1]|EventB”
EAI	“Dynamic[1]|Graft[0]|EAI”
Dynamic	“Dynamic”
Graft	“Dynamic[1]|Graft”
SubCallA	“SubCallA”
SubCallB	“SubCallA|SubCallB”

If the outstanding item is in the main procedure, the ProcPath string will simply consist of the name of the step for that outstanding item (see the NormalA step in the example above).

If the outstanding item is located in a sub-procedure, the ProcPath string will consist of the name of each sub-procedure call step leading to that outstanding item, followed by the step name, each separated by a vertical bar (see SubCallB in the example).

If the case family contains dynamic sub-procedure call steps or graft steps that start multiple sub-procedures (see the Dynamic and Graft steps in the example), the name of the dynamic sub-procedure call step and graft step in the ProcPath will include a **StartIndex** in square brackets. The StartIndex (which is zero based) indicates the sequential order in which the sub-procedure was started by the engine for that dynamic sub-procedure call step or graft step. It is used in the ProcPath to be able to identify the path through multiple sub-procedures to the desired outstanding item.

Triggering Events

An Event step is a step that is processed by calling the **triggerEvent** method on sCaseManager. You can call triggerEvent for a particular Event step:

- before the process flow has reached the Event step,
- after the process flow has halted at the Event step, or
- after the Event step has been processed.

When the triggerEvent method is called, the process flow will proceed from the Event step in the procedure.

You can also call triggerEvent on the same Event step multiple times. This reactivates that portion of the procedure each time it is called.

When an Event step is triggered with the triggerEvent method, you can optionally pass vField objects that identify fields whose case data you want to update. You can also specify that the new data overwrite both “case data” and “work item data”. See “Case Data vs. Work Item Data” on page 166 for information about the difference between case data and work item data. (Note that if you are updating case data when performing a triggerEvent operation, the fields you want to update may reside in a sub-case. For details about identifying the fields in the sub-case, see the triggerEvent method in the on-line help system.)

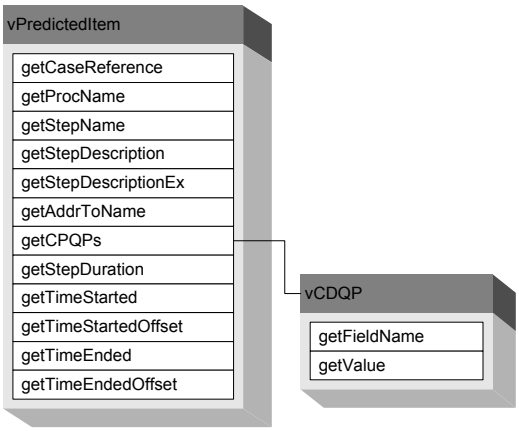
The triggerEvent method also allows you to “resurrect” (make active again) a closed case by passing True in the *Resurrect* parameter.

The triggerEvent method can also be used to dynamically recalculate deadlines. For information, see “Dynamically Recalculating Deadlines” on page 128.

See the on-line help for syntax details about the triggerEvent method.

Predicting Cases

Case prediction provides the means for predicting the expected outcome of an actual or an imaginary case. Running a case prediction function causes a list of "predicted work items" (**vPredictedItem** objects) to be returned that represent the work items that are currently due (outstanding work items), as well as the work items that are expected to be due in the future.



Note - To be able to use case prediction functions, you must be using a TIBCO iProcess Engine.

Included with the work items returned is information about the expected times the work items are predicted to start and end, providing information that can be used to predict the outcome of the case. This can be used to improve work forecasting and estimate the expected completion of cases.

The prediction process moves through the designated procedure(s) step-by-step using live or simulated case data to decide which path to take in the procedure. Each step uses an expected duration (decided at design time) to calculate a start and end processing time for each step as it progresses through the prediction process. When the process is complete, you can use these end processing times to predict the outcome of the case(s).

During the prediction process, initial and release scripts are executed, deadlines are processed, and withdrawal actions are performed, where appropriate.

There are two primary types of case prediction:

- *Background case prediction*, which allows you to predict the outcome of all active cases across all (enabled) procedures on the node. This type of case prediction is performed using the **getPredictedItemList** method. See "Background Case Prediction" on page 87 for more information.
- *Ad-hoc case prediction*, which is sub-divided into the following two operations:
  - *Live case prediction*, which allows you to predict the process path and duration of an active case. This type of case prediction is performed using the **predictCase** method. See "Live Case Prediction" on page 88 for more information.
  - *Case simulatio*n, which allows you to simulate the processing of an imagined case (with simulated case data), to predict its expected outcome. This type of case prediction is performed using the **simulateCase** method. See "Case Simulation" on page 88 for more information.

Defining Case Prediction

The following subsections describe how case prediction is defined for a procedure using TIBCO Business Studio.

Step Duration

As you are defining a procedure using TIBCO Business Studio, a "duration" is defined for each step in the procedure. The duration is the expected time interval between when the work item will become "active" and when it will be released. This is defined on the **Deadline/Step Duration Definition** dialog in TIBCO Business Studio. In TIBCO iProcess Server Objects, the step's duration is represented by a **vDuration** object, which is returned by the **getDuration** method on each of the following step objects: vNormalStep, vEventStep, vEAIStep, vSubProcCallStep, vDynamicSubProcCallStep, and vGraftStep.

The vDuration object contains methods that allow you to determine the duration definition that was specified when the step was defined:

- **getType** - This returns the type of duration that is defined for the step. It may be defined as: no duration, a duration expression (dynamic), a duration period (static), or that the deadline defined for the step be used as its duration. These are defined in the **SWDurationType** enumeration.
- **getDurationValues** - This method returns a list of **vDurationValue** objects, which provide access to the values that were entered in the Deadline/Step Duration Definition dialog for that step.

Note that if the step deadline is used for the duration (**vDuration.getType = swDurationDeadline**), the duration value will NOT be returned by the **vDurationValue.getValue** method. Rather, it will be returned by the **vDeadlineValue.getValue** method.

On the **Deadline/Step Duration Definition** dialog, you can also specify that the duration definition be used in the prediction calculation, but to exclude the step (work item) from the list of work items that are returned when case prediction is performed. This option allows you to specify that only the work items that are processed manually be included in the prediction output, excluding "broker type" steps and EAI steps. The setting of this option is available with the **isPrediction** method on each of the step objects that can be used in case prediction (vNormalStep, vEventStep, vEAIStep, vSubProc-CallStep, vDynamicSubProcCallStep, and vGraftStep) — the isPrediction method returns True if the step is excluded from the prediction results, but the step's duration definition is still used in the prediction calculation.

The **Procedure Status** dialog also includes a **Duration** button, which displays a dialog that allows you to set a duration for the procedure. This is intended to be used to assign a duration to a sub-procedure, allowing you to assign a duration for the entire sub-procedure rather than each step in the sub-procedure. If a duration is assigned for the sub-procedure, it takes precedence over a duration that is defined for the sub-procedure call step. If a duration is defined for the sub-procedure, it is accessible with the **vProcDef.getDuration** method.

Conditional Actions for Case Predictions

The "conditional" definition for a step in TIBCO Business Studio includes a "predicted condition." When a "conditional action" is encountered in a step as the prediction process moves from step to step, the "prediction condition" specifies how the conditional action is to be handled by the prediction process. It can handle it in one for the following three ways:

- **Evaluate** - The conditional expression will be evaluated to determine the path to take.
- **Default to True** - The conditional expression will default to True.
- **Default to False** - The conditional expression will default to False.

The definition given the "predicted condition" in TIBCO Business Studio is available with the **vConditional.getPredictedCondition** method. This method returns an enumeration constant (**SWConditionPredictType**) that identifies how the "prediction condition" was defined in TIBCO Business Studio.

Performing Case Prediction

As stated earlier, there are two primary types of case prediction: background and ad-hoc. Ad-hoc case prediction is further sub-divided into live case prediction and case simulation. The methods of performing each of these types of case prediction are described below.

Background Case Prediction

Background case prediction allows you to predict the outcome of all active cases across all (enabled) procedures on a node (see below for information about enabling a procedure for background prediction). This type of case prediction is performed by calling the **getPredictedItemList** method on **sCaseManager**. The getPredictedItemList method returns a pageable list of **vPredictedItem** objects, one for each current and future outstanding work item, for all active cases, on all procedures on the node (for which prediction is enabled).

A procedure must be *enabled* to take part in a background case prediction operation. When a procedure is defined using TIBCO Business Studio, the **Prediction** flag on the **Procedure Status** dialog must be checked to enable prediction on this procedure. This flag can be accessed with the **isPrediction** method on **vProcDef**. This method returns True if prediction has been enabled for the procedure.

The vPredictedItem objects returned by the getPredictedItemList method can be filtered and/or sorted by passing in a **vPredictionCriteria** object with the method call. The vPredictionCriteria object contains filter and sort criteria for the predicted items. See "Filtering and Sorting Predicted Items" on page 90 for more information. You can also filter or sort on case data in CDQP fields in the prediction results, provided the CDQP fields have been configured for prediction (see "Including Case Data Queue Parameter Data in Prediction Results" on page 89 for more information).

You can also *persist* the pageable list returned by the getPredictedItemList method. This is done by saving the "HeldId" for that pageable list, then using that HeldId as an input parameter on the **getPredictedItemListHeld** method at a later time. This returns that same pageable list of vPredictedItem objects as the original call to getPredictedItemList.

Live Case Prediction

Live case prediction allows you to predict the process path and duration of an active case. This type of case prediction is performed by calling the **predictCase** method on **sCaseManager**. The predictCase method returns an array of **vPredictedItem** objects, one for each work item that was outstanding when the method was called, and one for each work item that is expected to be outstanding in that case in the future.

Note that the procedure from which the live case was started does not need to have case prediction enabled in TIBCO Business Studio (i.e., the **Prediction** flag on the **Procedure Status** dialog does not need to be set) to run a case prediction operation on the live case.

Case Simulation

Case simulation allows you to simulate the processing of an imagined case. This type of case prediction allows you to provide case data that is used in the simulation of an imagined case. The case data is used to decide which path to take when there are conditional actions in the steps of the procedure. This type of case prediction is performed by calling the **simulateCase** method on **sProcManager**.

This method returns an array of **vPredictedItem** objects, one for each work item that is expected to be processed in the imagined case.

The simulateCase method provides a *StepNames* parameter that allows you to specify the step(s) from which the simulated case is to start. If omitted, the simulation starts at the start step of the procedure.

Note that the procedure on which you want to run a case simulation does not need to have case prediction enabled in TIBCO Business Studio (i.e., the **Prediction** flag on the **Procedure Status** dialog does not need to be set).

Sub-Procedures, Dynamic Sub-Procedures, and Graft Steps in Prediction

The following summarizes the way in which the prediction methods handle sub-procedure call steps, dynamic sub-procedure call steps, and graft steps.

Sub-Procedure Call Steps

- **predictCase and getPredictedItemList**
  - vPredictedItem objects will be returned for sub-procedure call steps that have not yet been processed (the process flow has not reached the step).
  - vPredictedItem objects will NOT be returned for sub-procedure call steps that are currently outstanding (the process flow has reached the step and its sub-procedure has been started). However, a vPredictedItem object will be returned for each outstanding item in the sub-procedure that has been started by the sub-procedure call step.
- **simulateCase**
  - vPredictedItem objects will NOT be returned for sub-procedure call steps. However, a vPredictedItem object will be returned for each of the outstanding steps predicted to be outstanding in the sub-procedure started by the sub-procedure call step of the simulated case.

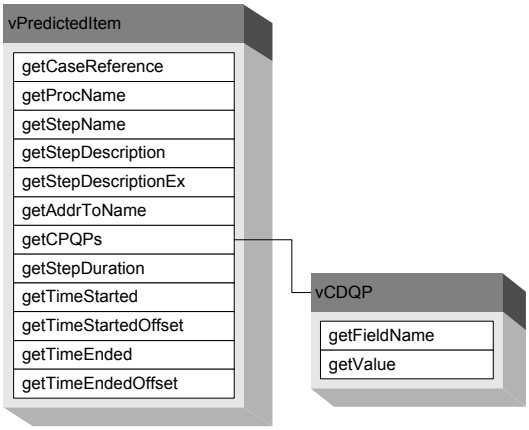
Dynamic Sub-Procedure Call Steps and Graft Steps

- **predictCase and getPredictedItemList**
  - vPredictedItem objects will be returned for dynamic sub-procedure call steps and graft steps that have not yet been processed (the process flow has not reached the steps).
  - vPredictedItem objects will NOT be returned for dynamic sub-procedure call steps nor graft steps that have been processed (the process flow has reached the steps). Also note that vPredictedItem objects will NOT be returned for any outstanding items in sub-procedures that are started by dynamic sub-procedure call steps or graft steps — that's because the prediction operation only looks at the definition of the procedure; it does not look at the contents of the array field to determine which sub-procedures have been started by the dynamic sub-procedure call step or graft step.
- **simulateCase**
  - vPredictedItem objects will be returned for dynamic sub-procedure call steps and graft steps predicted to be outstanding in the simulated case. Note, however, that since it's a simulated case, no sub-procedures are actually started by dynamic sub-procedure call steps or graft steps, therefore there can be no predicted items for sub-procedures started by those steps (and there is no means to simulate starting sub-procedures from dynamic sub-procedure call steps and graft steps).

Including Case Data Queue Parameter Data in Prediction Results

When Case Data Queue Parameter (CDQP) fields are defined with the **swutil** utility, an attribute may be set specifying whether or not the CDQP field is to be included in prediction results (it defaults to False). The setting of this attribute is available with the **isPrediction** method on **vCDQPDef** for each CDQP field.

If this attribute is set to True for a specific CDQP field, and that CDQP field is encountered in the prediction process, a **vCDQP** object representing the CDQP field is returned on the vPredictedItem object. All CDQP fields returned are available from vPredictedItem with the **getCPQPs** method.



Note that certain conditions must be met for CDQP fields to be returned in the prediction results. They are:

- Only CDQP fields that have prediction enabled in their definition will be returned.
- vCDQP objects are returned only on vPredictedItem objects that represent normal steps.
- vCDQP objects are returned only if the CDQP field is defined on the work queue for the user / group who is the addressee of the step.
- If the simulateCase method is called, CDQP fields will NOT be returned on work items if the addressee of the step is a variable (such as sw_starter). They will be returned if the addressee is a role or has been defined explicitly. (CDQPs will be returned on work items if the addressee is a variable when predictCase or getPredictedItemList is called.)

The vCDQP objects returned by the getCPQPs method provide access to the case data in those CDQP fields, as well as the ability to filter and sort on that case data when prediction is run using the getPredictedItemList method (which returns a pageable list; filtering and sorting can't be performed on the vPredictedItem objects that are returned by the predictCase and simulateCase methods).

Filtering and Sorting Predicted Items

You can filter and/or sort the results of the **getPredictedItemList** method, which are returned in a pageable list. This is done by passing a **vPredictionCriteria** object in the method call. Note that because predicted items are stored in the database, filtering them works in the same way as filtering cases when you have the database case filtering enhancement; this includes the limitation that the only special characters that can be used when using regular expressions are '*' and '?' — these work as wildcard characters, where the asterisk matches zero or more of any character, and the question mark matches any single character. See the *Filtering Work Items and Cases* chapter on page 247 for more information. Also see the *Sorting Work Items and Cases* chapter on page 273 for information about how to sort items that are returned in a pageable list.

The *Filtering Work Items and Cases* and the *Sorting Work Items and Cases* chapters provide lists of system fields that can be used when filtering / sorting. Note, however, that when filtering and sorting **vPredictedItem** objects, you are restricted to using the system fields listed in the table below. The "Filter" and "Sort" columns indicate whether you can filter or sort using that system field.

System Field	Filter	Sort	Data Type
SW_STEPNAME	X	X	String
SW_STEPDESC	X	X	String
SW_STEPDESC2	X	X	String
SW_CASENUM	X		Numeric
SW_MAIN_CASENUM	x		Numeric

System Field	Filter	Sort	Data Type
SW_PARENT_CASENUM	X		Numeric
SW_PRONUM	X		Numeric
SW_PARENT_PROCNUM	X		Numeric
SW_CASEREF		X	Numeric
SW_STEPDURN_SECS	X		Numeric
SW_STEPDURN_USECS	X		Numeric
SW_ADDRESSEE	X	X	String
SW_ARRIVALDATE	X		String (“YYYY-MM-DD HH:MM:SS”)*
SW_ARRIVALDATE_USECS	X		Numeric
SW_LEAVE_DATE	X		String (“YYYY-MM-DD HH:MM:SS”)*
SW_LEAVE_DATE_USECS	X		Numeric

* The hour component is in 24-hour format.

You can also filter on CDQP fields that are in the predicted work items returned by getPredictedItem-List, as long as they have been configured for case prediction.

If no sort criteria are specified, the arrival date and time are used to sort the predicted items.

Note - When filtering predicted items, if an invalid system field is used in your filter expression, an error is not produced. Instead, the filter operation will not return any items.

Using Graft Steps

A "graft step" allows an external application to inform the client application at *run-time* how many "tasks" it intends to "graft" to that step. For each task that it will graft to the graft step, the external application can specify any number of sub-procedures and/or external processes to start. You can graft multiple tasks to one graft step.

For example, a financial application determines that a credit check and a transfer of funds are required as part of the main procedure. When another case is started, it determines that only a transfer of funds is required. This means that the procedure is dynamic and cannot be decided at procedure definition time. One of the processes is a sub-procedure and the other is an external process run by the financial system. They can be specified as a task at run-time and attached to a graft step for processing.

A graft step is considered complete (i.e., its "release" actions are processed), when:

- It has been processed as an action of another step (i.e., the process flow has reached the graft step), ***and***
- the application has informed the graft step how many tasks it needs to complete (i.e., the graft step’s task count has been set — **isTaskCntSet** = True), ***and***

- the graft step's task count (**getTaskCnt**) has reached zero (the task count is decremented when a task has been started or if you delete a task), ***and***
- all of the sub-procedures and/or external processes started for the graft step have completed.

Defining Graft Steps

Graft steps are defined using TIBCO Business Studio. A graft step is represented by the **vGraftStep** object. The following are elements of a graft step definition:

- **Sub-Procedures / External Processes to Start** - When a graft step is defined in TIBCO Business Studio, you do not specify the names of sub-procedures and/or external processes that will be started for the graft step; those are specified at run-time by the external application. Instead, you specify a text "array field". At run-time, the external application will write the names of the sub-procedures and external processes to start into the elements of the array field.

  The **vGraftStep.getSubProcNameFld** method returns the name of the array field that was specified when the graft step was defined.

  See "Array Fields" on page 172 for information about how array fields are used with graft steps.
- **Return Status** - When a graft step is defined in TIBCO Business Studio, you can specify a numeric array field, whose elements will contain a return status for each corresponding sub-procedure and/or external process in the SubProcNameFld array field. See "Return Statuses" on page 95 for information about the return status.

Starting a Graft Task

A graft task is started with the **startGraftTask** method on **sCaseManager**. This method call requires that you first construct a **vGraftSubTask** object for each sub-procedure and/or external process you want initiated from the graft step. You must then pass an array of vGraftSubTask objects in the startGraftTask method call. The names of the sub-procedures and external processes that are being initiated from the graft step are written to elements of the array field that was specified in the graft step definition.

- If **sub-procedures** are started with the startGraftTask method, the engine will keep track of which sub-procedures have completed.
- If **external processes** are started with the startGraftTask method, the customer application must keep track of when each of the external processes has completed. The application must inform the engine by calling the **externalGraftProcessComp** method for each external process as it completes. (Note that calling the startGraftTask method does not actually start an external process. It is merely providing the names of external processes that must be completed before the graft task is considered complete.)

Calling the startGraftTask method causes the graft step's task count to be decremented by one. See "Setting the Task Count" below for more information.

When a graft task is started with the startGraftTask method, a "graft ID" must be specified. This ID is user-supplied and must be unique to this instance of the graft step. All other methods that can impact this graft step must include this ID to ensure they are impacting the appropriate graft step. The graft ID may be initially established with either the **startGraftTask** or the **setGraftTaskCnt** method, as either one may be the first method called for a particular instance of a graft step. (You could establish the graft ID for a graft step by calling the **deleteGraftTask** method first, but it really doesn't make any sense to delete a graft task before starting one or setting the count.)

Note that when sub-procedures are started with the **startGraftTask** method, the sub-procedures are started with the same precedence at which the parent case was started. That is, you cannot specify a separate sub-procedure precedence for sub-procedures started with the **startGraftTask** method.

For any given graft step, the startGraftTask method can be called multiple times (using the same graft ID). The sub-procedures and/or external processes specified in each subsequent call are appended to the existing sub-procedure / external process names in the array field in the graft step definition.

Case data may also be supplied when calling the startGraftTask method. This allows you to specify field values to be passed to sub-procedures that are started by the startGraftTask method. To pass case data, you must construct an array of **vField** objects that identify the fields/values, then include those objects when constructing the **vGraftSubTask** that identifies the sub-procedure to start when calling the startGraftTask method.

The field names should be preceded by the index into the array of sub-cases being grafted. Normally, only one case is grafted at a time, in which case, all the sub-case field names should be preceded by '$1|', for example:

```
$1|ADD_TIME_VALUE,24
$1|ALERTID,22
$1|BRANCHNAME,RSSI_GIMS
$1|CONFIDENTIALITY,C1
$1|DESTRSSI,RSSI_GIMS
$1|RISKID,9
$1|SEVERITY,1
$1|SRT_NUMBER,RIS-[RSSI_GIMS]-[SRT[06]-20]
```

If two cases were being started, you might have the following data:

```
$1|ADD_TIME_VALUE,24
$1|ALERTID,22
$1|BRANCHNAME,RSSI_GIMS
$1|CONFIDENTIALITY,C1
$1|DESTRSSI,RSSI_GIMS
$1|RISKID,9
$1|SEVERITY,1
$1|SRT_NUMBER,RIS-[RSSI_GIMS]-[SRT[06]-20]
$2|ADD_TIME_VALUE,26
$2|ALERTID,33
$2|BRANCHNAME,RSSI_DIMS
$2|CONFIDENTIALITY,C2
$2|DESTRSSI,RSSI_GIMS
$2|RISKID,7
$2|SEVERITY,2
$2|SRT_NUMBER,RIS-[RSSI_DIMS]-[SRT[06]-20]
```

Setting the Task Count

The customer application must inform the engine how many tasks must be completed for this graft step for it to be considered complete.

Setting the graft step's task count is done with the **setGraftTaskCnt** method. Calling this method causes the **isTaskCntSet** flag to be set to True, and increments the graft step's task count (available with **getTaskCnt**) by the number specified in the setGraftTaskCnt method. (Note that the task count may be negative if you start one or more tasks before you call the setGraftTaskCnt method.)

Note that this count is the number of "tasks" that have to be completed for the graft step to be released; it is *not* the number of sub-procedures or external processes that are being started by the graft task — each graft task can start multiple sub-procedures and external processes.

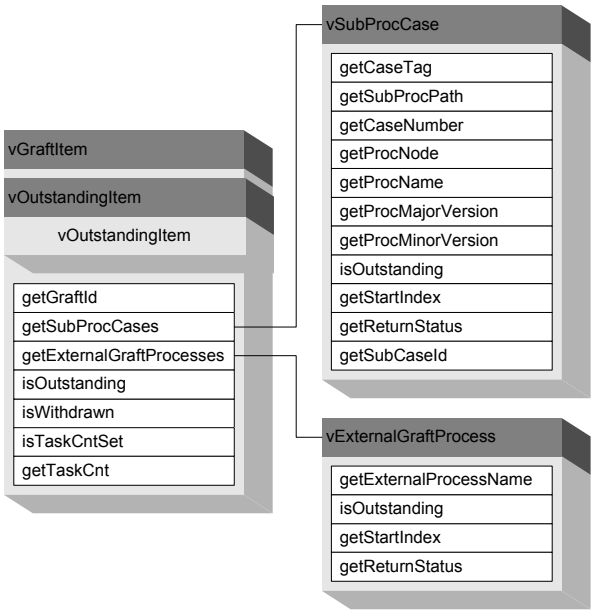
The task count is automatically decremented by one each time the **startGraftTask** or **deleteGraftTask** method is called. So the current task count tells you the number of tasks that are remaining to be completed.

Outstanding Graft Items

A graft step becomes "outstanding" when the graft task is initiated either with the startGraftTask or setGraftTaskCnt methods, or the process flow has reached the graft step.

When the graft step becomes outstanding, a **vGraftItem** object representing the outstanding graft item will be returned from **getOutstandingItems** on sCaseManager. (The getOutstandingItems method actually returns vOutstandingItem objects — the vGraftItem object is derived from vOutstandingItem.)

Note that although the graft item is considered "outstanding" (because it is returned when you call getOutstandingItems), the **isOutstanding** method on vGraftItem returns True only if the process flow has reached the graft step in the procedure. It does not return True if the graft item is outstanding because it was initiated with startGraftTask or setGraftTaskCnt.



You can determine the sub-procedures and/or external processes that were started by the outstanding graft item by calling the following methods on **vGraftItem**:

- **getSubProcCases** - Returns an array of **vSubProcCase** objects, one for each sub-procedure that was started by the graft item. This tells you ALL sub-procedures that were started, whether they have completed or not. You can determine whether or not the sub-procedure has completed by calling the **isOutstanding** method on vSubProcCase; this method returns True if the sub-procedure is still outstanding (it hasn't completed yet).

  See "The Sub-Case Object" on page 56 for more information about vSubProcCase objects.

- **getExternalGraftProcesses** - Returns an array of **vExternalGraftProcess** objects, one for each external process that was initiated by the graft item. This tells you ALL external processes that were initiated by the graft step, whether they have completed or not. You can determine whether or not the external process has completed by calling the **isOutstanding** method on vExternalGraftProcess; this method returns True if the external process is still outstanding (hasn't completed yet). It is flagged as completed when the application calls the **externalGraftProcessComp** method.

Return Statuses

A “return status” is available that provides status information for the sub-procedures and external processes that are initiated/completed for a graft step. This status is available by calling the **getReturnStatus** method on **vSubProcCase** (for sub-procedures) or **vExternalGraftProcess** (for external processes). These are described below:

- **Sub-Procedures** - The **getReturnStatus** method on **vSubProcCase** returns an integer that indicates the current status of the sub-procedure. The status for sub-procedures is set by the system.

  The integers returned by getReturnStatus are defined in the **SWSubProcStatusType** enumeration, as follows:

SWSubProcStatusType	
swNoAttempt	0
swStarted	1
swCompleted	2
swErrSubProc	-1
swErrTemplate	-2
swErrInTemplateVer	-3
swErrOutTemplateVer	-4

- **External Processes** - The return status is only applicable to external processes after they have completed.

  When the **externalGraftProcessComp** method is called to tell the engine that the external process is complete, a user-defined status can be specified in the **vGraftSubTask** object that is passed in the externalGraftProcessComp method call. This *Status* parameter can be any integer the user chooses.

  The **getReturnStatus** method on **vExternalGraftProcess** returns the integer that was specified when the externalGraftProcessComp method was called.

*Note - You can also get the return status by using the ReturnStatus array field (**getReturnStatusFld**) that was specified in the graft step definition (**vGraftStep**). The elements of the ReturnStatus array field contain the status codes for the corresponding sub-procedures or external processes named in the SubProcNameFld (**getSubProcNameFld** on vGraftStep).*

Deleting a Task

After starting a graft task with the startGraftTask method, you may decide you don't want to complete that task and would like to delete it from the list of tasks the graft step needs to complete. This can be done by calling the **deleteGraftTask** method.

Calling deleteGraftTask causes the task count (**getTaskCnt**) to be decremented by one.

Completing a Graft Step

A graft step is considered complete, and is subsequently released, when:

- the process flow has reached the graft step — **vGraftItem.isOutstanding** = True
- the graft step's task count has been set — **vGraftItem.isTaskCntSet** = True
- the graft step's task count has reached zero (the task count is decremented when a task has been started or if you delete a task) — **vGraftItem.getTaskCnt** = 0
- all of the sub-procedures and/or external processes started for the graft step have completed (the engine keeps track of when the sub-procedures have completed; the customer application must inform the engine when each external process has completed by calling the **externalGraftProcessComp** method)

When all of these conditions are met, the release actions for the graft step are executed.

Error Processing

Graft step definitions provide options that allow the definer to specify whether or not to halt processing if an error is encountered during processing of the sub-procedures that are started by the graft step. These options are available with the following methods on **vGraftStep**:

- **isHaltOnSubProc** - Returns True if processing should be halted when the "sub-procedures to start" array field contains elements that specify non-existent sub-procedures.
- **isHaltOnTemplate** - Returns True if processing should be halted when the "sub-procedures to start" array field contains elements that specify sub-procedures that do not use the same parameter template. (Parameter templates are used when defining procedures to ensure that the same input and output parameters are used when starting multiple sub-procedures from a graft step; for information about parameter templates, see TIBCO Business Studio documentation.)
- **isHaltOnTemplateVer** - Returns True if processing should be halted when the "sub-procedures to start" array field contains elements that specify sub-procedures that do not use the same version of parameter template.

These options for halting processing on specific error conditions have the following affects:

Errors during <u>initial processing</u> (when the graft step is processed as an action of another step):

- If an error is encountered and the step is <u>defined to halt</u>:
  - The message that resulted in the error will be retried the number of times specified in the engine. (This is specified with a background attribute: IQL_RETRY_COUNT = the number of times the message will be retried; IQL_RETRY_DELAY = the number of seconds between retries.) If the message retries do not result in a successful initial processing, the following apply:
    - Processing of the entire step is halted at this point -- it will always be left "waiting" for the sub-case that's in error to be completed.
    - All sub-procedures that have been started from the step are rolled back.
    - An SW_ERROR message is logged stating the reason for the failure.
    - An appropriate entry is written to the audit trail for the parent case.

- If an error is encountered and the step is defined to NOT halt:
  - The other valid sub-procedures specified in the SubProcName array field will be started as usual.
  - An SW_WARN message is logged stating the reason for the failure.
  - An appropriate entry is written to the audit trail for the parent case.

Errors during completion processing of one of the sub-cases:

- If an error is encountered and the step is defined to halt:
  - The message that resulted in the error will be retried the number of times specified in the engine. (This is specified with a background attribute: IQL_RETRY_COUNT = the number of times the message will be retried; IQL_RETRY_DELAY = the number of seconds between retries.) If the message retries do not result in a successful completion processing, the following apply:
    - Processing of the entire step is halted at this point -- it will always be left "waiting" for the sub-case that's in error to be completed.
    - The "sub-case completed" transaction for the sub-case in error is aborted -- this does not cause transactions from other valid sub-case completions to be aborted.
    - An SW_ERROR message is logged stating the reason for the failure.
    - An appropriate entry is written to the audit trail for the parent case.
- If an error is encountered and the step is defined to NOT halt:
  - The "sub-case completed" transaction for the sub-case in error is ignored (including returned output parameter data).
  - The status of the sub-case is set to "complete" so that the step can be released when all other sub-cases complete.
  - An SW_WARN message is logged stating the reason for the failure.
  - An appropriate entry is written to the audit trail for the parent case.

Note that if none of the “halt on” selections are selected in TIBCO Business Studio when the graft step is defined, and one of the error conditions are encountered (e.g., sub-procedures using different templates), the process will continue, which could possibly result in errors in case data.

Transaction Control Steps

Transaction control steps provide a mechanism, within a procedure, to allow more transaction granularity within a sequence of EAI steps (transaction control steps can *only* be used in conjunction with EAI steps).

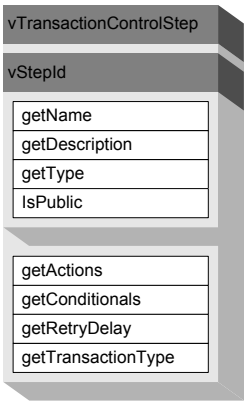
By default, the Background process groups a series of connected EAI steps into one transaction. If a failure occurs in any EAI step in the series, the entire transaction is rolled back. A *transaction control step* can be placed within the series of EAI steps to break the transaction into multiple transactions. When the process flow reaches the transaction control step, the current transaction can be either committed and a new transaction started, or the current transaction can be aborted, depending on how the transaction control step has been defined in TIBCO Business Studio.

For information about creating transaction control steps, see the TIBCO Business Studio documentation.

The vTransactionControlStep Object

The definition of a transaction control step is represented by the **vTransactionControlStep** object.

This object is available by using either **getSteps** or **getStepIds** to get **vStepId** objects, then use **getType** to identify the step as a transaction control step (**getType** = **swTransactionControl**), then cast it to a **vTransactionControlStep** object.



Type of Transaction Control Step

When a transaction control step is defined in TIBCO Business Studio, it is configured to be one of the following three types:

- **Commit and Continue** - This type specifies that the current transaction be committed, and that a new transaction be started for subsequent steps using the same Background process. The benefit of choosing this option is that it is faster, as the same process starts the new transaction.
- **Commit and Concede** - This type specifies that the current transaction should be committed, and that a new transaction be started for subsequent steps, except a *different* Background process will be used for the second transaction.

  The Background process processes the first transaction and updates the database. It then sends a message back to the Mbox where the messages are stored. Processing of the next transaction proceeds when the Background process (either the same one that processed the first transaction, or another one) reads the message from the Mbox and processes it. The benefit of choosing this option is that it enables load balancing because a different Background process can process the second transaction.
- **Abort** - This option causes the abortion and rollback of the current transaction when the process flow reaches the transaction control step. This option is typically used with a Conditional. This allows you to specify a condition on which the transaction should be rolled back. After the

aborted transaction is rolled back, it will be retried using the normal Mbox queue message retry behavior.

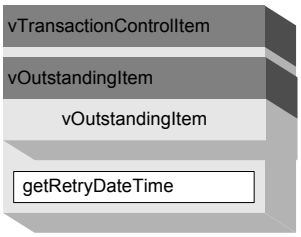
This type of transaction control step can be useful where a mixture of transactional and non-transactional EAI steps are used in a procedure. The Conditional can check for an error state, a corrective action can be performed in the non-transactional EAI step, then the abort transactional control step will roll back the transaction.

The transaction control step type can be accessed using the **getTransactionType** method on the **vTransactionControlStep** object. This method returns one of the following constants:

Constant	Description	Value
swContinue	Commit and continue transaction control step.	'C'
swConcede	Commit and concede transaction control step.	'D'
swAbort	Abort transaction control step.	'A'

These constants are defined in the **SWTransactionType** enumeration.

Outstanding Transaction Control Steps



The **vTransactionControlItem** object represents an "outstanding" transaction control step. A transaction control step becomes outstanding when the process flow reaches the step. The transaction control step is no longer outstanding when the transaction controlled by the transaction control step is either committed or aborted.

The **getOutstandingItems** method on **sCaseManager** returns an array of **vOutstandingItem** objects, one for each outstanding step in the case family. Use the *aOutstandingItemContent* parameter to specify that you would like outstanding transaction control steps returned (set *isWithTransactionControlItems* to True).

See "Determining Outstanding Items" on page 82 for more information about obtaining and using outstanding items.

Retrying Failed Transactions

If a transaction that is controlled from a "commit and continue" transaction control step fails, the failed transaction will be retried after a specified period of time. The following methods are provided to reflect this retry delay/time:

- **vTransactionControlStep.getRetryDelay** - This method returns the number of minutes in which a failed transaction will be retried (only applicable for "commit and continue" transaction control steps). This value is specified when the transaction control step is defined.
- **vTransactionControlItem.getRetryDateTime** - This method returns the date and time that the failed transaction will be retried. This date/time is calculated from the time the transaction failed, using the number of minutes specified when the transaction control step was defined (see **getRetryDelay** above).

If a transaction that is controlled from a “commit and continue” transaction control step fails for any reason, the Deadline Manager will retry the failed transaction in the period of time specified when the step was defined. The getRetryDateTime method will return the date/time of this one-time retry. This one-time retry by the Deadline Manager is for the purpose of allowing the reason for the failure to correct itself so that the transaction can then be successfully processed.

Note, however, that if a failed transaction failed for some reason that is not corrected in the specified delay time, the transaction may be continually retried by the standard Mbox message queue process (the default is to retry 12 times, once every 5 minutes). These subsequent retries are not managed by the Deadline Manager. Therefore, the date/time returned by the getRetryDateTime method does not reflect these additional retries — it will only contain the date/time of the initial retry — the one controlled by the Deadline Manager.

Audit Trail Messages

The following are the messages that are written to the audit trail when an action is performed against a transaction control step. These are available with the **vAuditStep.getAction** method.

Constant	Description	Value
swTransProcessed	Transaction Control Step Processed	54
swTransStarted	Transaction Control Step - New Transaction Started	55
swTransRestart	Transaction Control Step - Retry Time Expired	56
swTransAborted	Transaction Control Step Processed - Transaction Aborted	87

These constants are defined in the **SWAuditActionType** enumeration.

Suspending Cases

You can suspend activity in a case family, which includes the main case and all of its sub-cases. This is done with the **setCaseSuspended** method on **sCaseManager**. Suspending a case family has the following effects:

- Normal Steps - You cannot lock any work items in the case family. If a work item is already locked when its state is changed to "suspended," the work item may still be released, and the release instructions will be carried out. Any new work items as a result of the release will immediately become suspended, unless they are flagged to ignore suspensions (described below). If the work item is kept, it will immediately become suspended, and it cannot be locked again until the suspended state is removed.
- Deadlines will continue to expire, however, if one expires while the case family is suspended, no action will be carried out until the suspended state is removed. Once the suspended state is removed, the process flow will proceed as usual.
- Event steps do not support case suspension, i.e., an event can be triggered on a case that is suspended. However, subsequent steps are suspended, unless they are flagged to ignore case suspension (described below).

- Withdrawals do not support case suspension. For example, if you suspend a case then trigger an event (see above), if the subsequent action is to withdraw an outstanding item, the withdrawal will be processed.

Note - You must be using a TIBCO iProcess Engine to use this functionality.

Setting the state of a case family to suspended causes the **isSuspended** flag on the **vACase**, **vNormalItem**, and **vWorkItem** objects to be set to True. Unsuspending the case family causes this flag to be set to False.

When calling the setCaseSuspended method, you can optionally pass **vField** objects to identify fields whose case data you want to update. You can also specify that the new data be written to both "case data" and "work item data" (using the *aUpdateOutstanding* parameter on setCaseSuspended). (See the on-line help system for syntax details.)

When a case is suspended, the following audit action (**SWAuditActionType**) is written to the audit log:

- **swSuspendedBy** - The case family is set to “suspended.”

Reactivating a Suspended Case

To reactivate a suspended case, call the **setCaseSuspended** method on **sCaseManager**, passing False in the *aSuspend* parameter.

When a suspended case is reactivated, the following audit action (**SWAuditActionType**) is written to the audit log:

- **swResumedBy** - The case family is set to “active.”

Ignoring Case Suspension

When a procedure is created with TIBCO Business Studio, normal, EAI, sub-procedure call steps, dynamic sub-procedure call steps, and graft steps can be flagged to ignore suspensions. If flagged to ignore suspensions, a step is processed normally even if the case in which it is located is suspended. This flag is reflected in the **isIgnoreCaseSuspend** flag on the **vNormalStep**, **vEAIStep**, and **vSubProcCallStep, vDynamicSubProcCallStep**, and **vGraftStep** object for the particular step.

Closing Cases

To close cases using any of the methods described here, the user must have system administrator authority (MENUNAME = ADMIN) — see "User Attributes" on page 187 for information about the MENUNAME attribute.

Active cases of a procedure can be closed using the following methods:

- **sCaseManager.closeCases** - This method allows you to close one or more cases of a specified procedure.
- **sCaseManager.closeCasesByCriteria** - This method allows you to close cases based on a filter criteria expression. Only those cases that match the criteria expression are closed. The filter criteria allowed is the same as the criteria used when filtering a list of cases. For information about the allowable filter criteria, see the appropriate *Filtering Work Items and Cases* chapter on page 193, page 221, or page 247.

The close cases methods also provide an optional *aDoEvent* parameter that when set to true causes a procedure-level event to be triggered when the cases are closed. The procedure can be defined to catch the event, then perform business logic either before or after the cases are closed. This parameter is only relevant when an iProcess Engine version 11.4.0 or newer is being used. For earlier versions, this parameter is ignored. Default = true.

The close cases methods also provide an optional *aPriority* parameter that can be used to increase the case close processing in the background. You can increase or decrease the priority at which cases are closed using the *aPriority* parameter. See the help system for details about the *aPriority* parameter.

After closing a case, the **vACase.isActive** method will return False to indicate it is no longer active.

Resurrecting a Closed Case

A closed case can be "resurrected" (i.e., its status changed to "active") using the **triggerEvent** method on sCaseManager. Call triggerEvent and pass True in the *aResurrect* parameter (you must be using a TIBCO iProcess Engine to use this parameter). The case will now become active, and the process flow will proceed from the Event step that was triggered by the triggerEvent method.

Purging Cases

To purge cases using any of the methods above, the user must have system administrator authority (MENUNAME = ADMIN) — see "User Attributes" on page 187 for information about the MENUNAME attribute.

Purging cases permanently deletes them from the system. You can purge cases using the following methods:

- **sCaseManager.purgeCases** - This method purges one or more cases.
- **sCaseManager.purgeCasesByCriteria** - This method purges cases based on a filter criteria expression. Only those cases that match the criteria expression are purged. The filter criteria allowed is the same as the criteria used when filtering a view or XList of cases. For information about the allowable filter criteria, see the appropriate *Filtering Work Items and Cases* chapter on page 193, page 221, or page 247.
- **sCaseManager.purgeAndReset** - This method purges ALL cases of the procedure and resets the case counter to 0 (**getCaseCnt** on vProcSummary and sCaseManager).

The purge cases methods also provide an optional *aDoEvent* parameter that when set to true causes a procedure-level event to be triggered when the cases are purged. The procedure can be defined to catch the event, then perform business logic before the cases are purged. This parameter is only relevant when an iProcess Engine version 11.4.0 or newer is being used. For earlier versions, this parameter is ignored. Default = true.

The purge cases methods also provide an optional *aPriority* parameter that can be used to increase the case purge processing in the background. You can increase or decrease the priority at which cases are purged using the *aPriority* parameter. See the help system for details about the *aPriority* parameter.

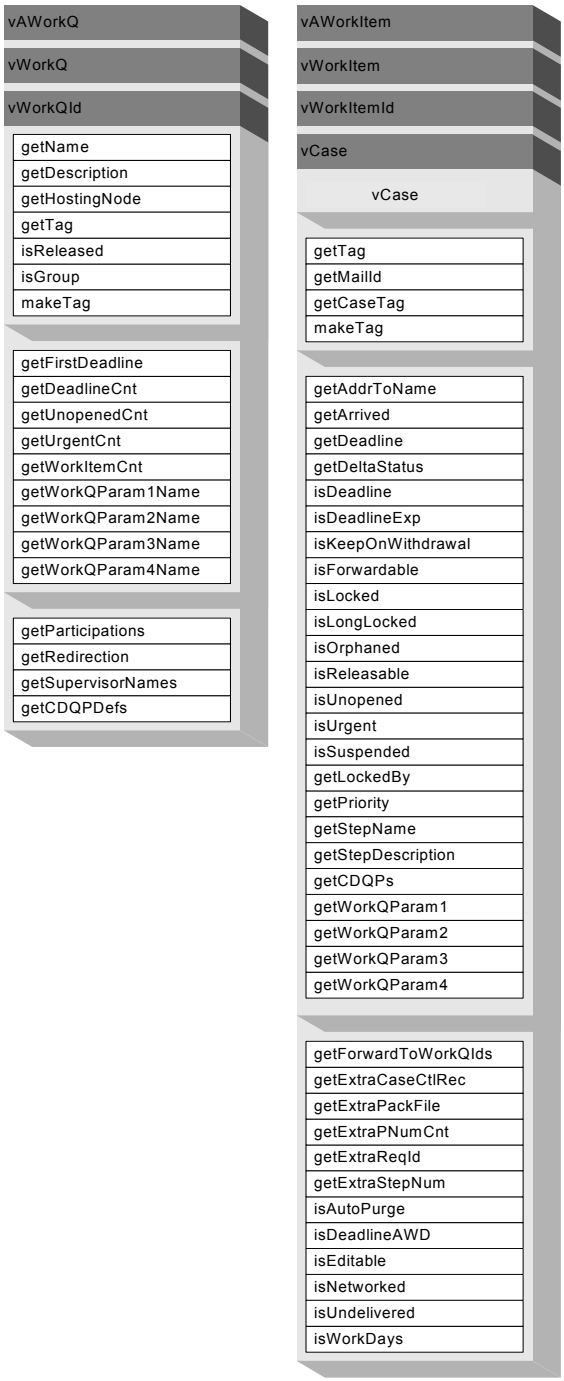
Both active and closed cases can be purged.

8

Managing Work Queues

Introduction

A work queue represents a list of work items that are awaiting action. A work queue can belong to an individual user or to a group of users. If it is a group work queue, all users that belong to that group have access to the work items in the work queue.



Work Queue Objects

Work queues are represented by the **vWorkQId**, **vWorkQ**, and **vAWorkQ** objects. These work queue objects provide access to information about a specific work queue, e.g., whether or not it's a group work queue (isGroup method), how many work items are in the work queue (getWorkItemCnt method), etc.

Work Item Objects

A work item represents an individual item in a work queue. Work items are represented by the **vWorkItemId**, **vWorkItem**, and **vAWorkItem** objects. These objects provide access to information about that specific work item, e.g., whether or not the work item is locked (isLocked method), the deadline for the work item (getDeadline), etc.

As work queue and work item objects are "Value Objects," they must be acquired by calling a method on a Server Object. The following subsections describe the methods used to get one or more work queue or work item objects from the server.

Test vs. Released Work Queues

Whenever a user or group is added, both a "test" and "released" work queue is automatically created for the user or group. Both of these work queues are returned when you retrieve lists of work queues (see the next section for information about retrieving work queues). Note, however, that neither the test

nor the released work queue for a user or group "officially" exists until a work item is sent to the work queue. If you attempt to access a work queue (either test or released) prior to it containing a work item, a "Queue not found" error is thrown.

Work items that result from cases of procedures that have a status of **swReleased** (see the **getStatus** method on vProc) are sent to the "released" work queue for the user or group that is the addressee of the step. Released work queues are accessible only by the user or group for whom the work queue was created, and by users that have been given "participation" access (see "Participating in Another User's Work Queue" on page 129).

Work items that result from cases of procedures that have a status of **swUnreleased** or **swModel** are sent to the "test" work queue for the user or group that is the addressee of the step. Note, however, that test work queues CANNOT be seen by the user or group that is the addressee of the step; they can only be seen by the user that started the case of the test procedure (which is the owner/definer of the unreleased procedure as only the owner/definer can start cases of an unreleased procedure). The test work queues are given the names of the addressees of the steps, but are not visible by those users. For instance, if user1 starts a case of an unreleased procedure, and the first step's addressee is user2, the work item is routed to user2's test work queue, but that queue is only visible by user1, not user2. This allows the definer of the procedure to ensure that the process flow is occurring as intended before releasing the procedure without having to log in and out as different users.

You can determine whether a work queue is a test or released queue by calling the **isReleased** method on vWorkQId (you can access vWorkQId only after a work item has been sent to the work queue). (The **isWorkQReleased** method is available on vNormalItem to determine if the outstanding item is located in a test or released work queue.)

Retrieving Work Queues

There are four Server Objects that contain methods used to retrieve work queues from the server:

- **sUser** - This Server Object is used when you want to work with a work queue from the perspective of a user. You can retrieve the work queues that the user is authorized to administer, or retrieve one or more work queues assigned to the user.
- **sWorkQ** - This Server Object is used to obtain a specific work queue that is identified in the **sWorkQ** constructor. Once you have this Server Object, you can perform functions such as retrieving work items from that work queue, processing work items (lock, keep, release, etc.), etc.
- **sWorkQManager** - This Server Object is used to perform administrative functions on one or more work queues.
- **sNode** - This Server Object is used to obtain a list of all work queues on the node.

The following table summarizes the methods available from these Server Objects for obtaining work queues from the server.

Server Object	Method	Retrieves from Server
sUser	getSupervisedQIds	An array of **vWorkQId** objects, one for each work queue that the user can supervise (for the purpose of defining participation and redirection schedules).
	getWorkQIds	An array of **vWorkQId** objects, one for each work queue for which the user is a member.
	getWorkQs	One or more specified **vWorkQ** objects.
sWorkQ	getWorkQ	A **vWorkQ** object for the work queue represented by the sWorkQ object.
sWorkQManager	getWorkQIds	An array of **vWorkQId** objects, one for each specified work queue.
	getWorkQIdList	A pageable list of **vWorkQId** objects, one for each work queue on the node.
	getWorkQs	An array of **vWorkQ** objects, one for each specified work queue.
	getWorkQList	A pageable list of **vWorkQ** objects.
	getWorkQListHeld	A pageable list of **vWorkQ** objects that had been previously held with the **hold** method.
	getAWorkQs	An array of **vAWorkQ** objects, one for each specified work queue.
	getAWorkQList	A pageable list of **vAWorkQ** objects.
	getAWorkQListHeld	A pageable list of **vAWorkQ** objects that had been previously held with the **hold** method.
sNode	getWorkQIdList	A pageable list of **vWorkQId** objects, one for each work queue on the node.
	getWorkQIdListHeld	A pageable list of **vWorkQId** objects that had been previously held with the **hold** method.

As you can see from the table above, the Server Object methods provide a variety of ways to retrieve **vWorkQId**, **vWorkQ**, and **vAWorkQ** objects. Which of these Value Objects are needed depends on the type of function you want to perform:

- **vWorkQId** - This object is typically retrieved so the work queue's tag can be obtained (with **getTag**). The tag is then used as an input parameter with another method (e.g, *aWorkQTag* is a parameter on the **changeParticipation** method), or as a parameter when constructing a Server Object (e.g., *aWorkQTag* is a parameter on the **sWorkQ** object constructor). (Note that a tag can also be constructed with the **makeTag** method if you know all of the components that make up the tag.)
- **vWorkQ** - This object provides access to additional information about the work queue, such as its first deadline, the number of unopened work items in the work queue, etc.
- **vAWorkQ** - This object provides access to the work queue's participation and redirection schedules, the names of users who can administer this work queue, and the definitions of Case Data Queue Parameter (CDQP) fields defined for the work queue. This information is typically used to administer the work queue.

Filtering Content when Retrieving Work Queues

When retrieving **vAWorkQ** objects from the server with the **getAWorkQs** or **getAWorkQList** methods, you can limit the amount of "content" that is returned from the server by using the *aWorkQContent* parameter on these methods.

```
vAWorkQ[] getAWorkQs(String[] aWorkQTags,
                     vAWorkQContent aWorkQContent)


sPageableList getAWorkQList(vAWorkQContent aWorkQContent,
                            int aItemsPerBlock)
```

The *aWorkQContent* parameter allows you to specify whether or not **vParticipation**, **vRedirection**, supervisor names (String objects), and **vCDQPDef** objects are also returned from the server with the **vAWorkQ** objects. To do this you must construct a **vAWorkQContent** object with the desired parameters (*aIsWithParticipation*, *aIsWithRedirection*, etc.) set to True, then pass that Value Object with your **getAWorkQs** or **getAWorkQList** method.

See "Retrieving Dependent Objects" on page 160 for more information about filtering content.

Retrieving Work Items

There are two Server Objects that contain methods used to retrieve work items from the server:

- **sUser** - This Server Object contains the **getWorkItems** method, which will return an array of **vWorkItems** based on the work item tags passed in the method.
- **sWorkQ** - This Server Object contains a number of methods that allow you to retrieve the work items from the work queue represented by the **sWorkQ** object.

The following table summarizes the methods available from these Server Objects for obtaining work items from the server.

Server Object	Method	Retrieves from Server
sUser	getWorkItems	An array of **vWorkItem** objects, one for each work item identified by the *aWorkItemTag* parameter.
sWorkQ	getWorkItems	An array of **vWorkItem** objects, one for each work item identified by the *aWorkItemTag* parameter.
	getWorkItemList	A pageable list of **vWorkItem** objects.
	getWorkItemListHeld	A pageable list of **vWorkItem** objects that had been previously held with the **hold** method.
	getAWorkItems	An array of **vAWorkItem** objects, one for each work item identified by the *aWorkItemTag* parameter.
	getAWorkItemList	A pageable list of **vAWorkItem** objects.
	getAWorkItemListHeld	A pageable list of **vAWorkItem** objects that had been previously held with the **hold** method.
	getAWorkItemListJMS	A pageable list that is used to obtain IDs for the purpose of obtaining work queue deltas via a JMS topic. For information, see "Work Queue Deltas Via a JMS Topic" on page 112.
	getAWorkItemListJMSHeld	A pageable list of **vAWorkItem** objects that had been previously held with the **hold** method. This method is used in conjunction with the **getAWorkItemListJMS** method (see above).

As you can see from the table above, the Server Object methods provide a variety of ways to retrieve **vWorkItem** and **vAWorkItem** objects. Which of these Value Objects are needed depends on the type of function you want to perform:

- **vWorkItem** - This object contains methods that are used to determine the work item's current state (**isLocked**, **isUnopened**, etc.), or to obtain the work item's tag (with **getTag**). The tag can be used as an input parameter on another method (e.g, *aWorkItemTags* is a parameter on the **lockItems** method). (Note that a tag can also be constructed with the **makeTag** method if you know all of the components that make up the tag.)
- **vAWorkItem** - This object provides access to information about the work item that would be used for administrative purposes.

*Note - There are no methods that return **vWorkItemId** objects. To use the methods on **vWorkItemId** (e.g., **getTag**), retrieve **vWorkItem** objects, which derive from **vWorkItemId**.*

Filtering and Sorting Work Items

The TIBCO iProcess Server Objects provide the ability to *filter* the arrays and lists of work items that are returned from the server. For instance, you may only want to return the work items that have a certain priority. This improves efficiency by retrieving only those work items needed. For information about filtering, see the appropriate *Filtering Work Items and Cases* chapter on page 193, page 221, or page 247.

You can also specify that the work items returned be sorted in a specific way. For instance, you may want all work items in the work queue to be sorted by priority, in ascending order. Details about sorting are provided in "Sorting Work Items and Cases" on page 273.

Filtering Content when Accessing Work Items

When retrieving **vWorkItem** objects from the server with the **getWorkItems** or **getWorkItemList** methods, you can limit the dependent objects that are also returned from the server. This is known as filtering "content". Filtering content is accomplished with the use of the *aWIContent* parameter on these methods:

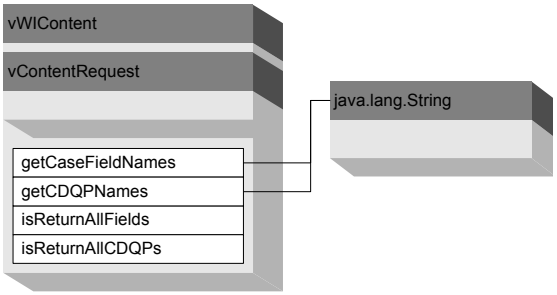
```
vWorkItem[] getWorkItems(String[] aWorkItemTags,
                         vWIContent aWIContent)


sPageableListR getWorkItemList(vWICriteria aWICriteria,
                               vWIContent aWIContent,
                               int aItemsPerBlock)
```

There are two types of content that can be filtered:

- **Case Data fields** - These represent the fields in the case. (Note that these fields and their respective data are not accessible from the **vWorkItem** object itself; rather they are accessible from the **vCase** object with the **getCaseFields** method — the **vWorkItem** object is derived from the **vCase** object.)
- **Case Data Queue Parameter (CDQP) fields** - These represent fields in work items that are used for filtering and sorting purposes. (See the "Using Case Data Queue Parameter Fields" section in the appropriate "Filtering Work Items and Cases" chapter on page 212, page 238, or page 264 for information about using CDQP fields for filtering, and "Using Case Data Queue Parameter Fields" on page 277 for information about using CDQP fields for sorting.)

To filter work items for content, you must construct a **vWIContent** object, then pass that object as a parameter with the **getWorkItems** or **getWorkItemList** method.



The following **vWIContent** parameters are used to filter Case Data fields:

- *aCaseFieldNames* - This parameter allows you to pass in an array of Strings identifying the Case Data fields you want returned with the work items.
- *aIsReturnAllFields* - This Boolean parameter allows you to specify that you want *all* Case Data fields returned with the work item (rather than having to list them all in the *aCaseFieldNames* parameter). Use the *aIsReturnAllFields* parameter with caution, however, as it can result in a significant amount of data being sent across the network.

The following **vWIContent** parameters are used to filter CDQP fields:

- *aCDQPNames* - This parameter allows you to pass in an array of Strings identifying the CDQP fields you want returned with the work items.
- *aIsReturnAllCDQPs* - This Boolean parameter allows you to specify that you want *all* CDQP fields returned with the work item (rather than having to list them all in the *aCDQPNames* parameter).

See "Retrieving Dependent Objects" on page 160 for more information about filtering for content.

Work Queue Deltas

The "work queue deltas" feature allows you to retrieve only the work items that have changed (i.e., the delta) on the work queue since the last refresh.

This feature allows you to determine the "delta count", i.e., the number of work items that have changed since the last refresh. You can use this count to determine whether you want to retrieve only the delta work items, or if you want to retrieve the entire list. For each of the retrieved work items, you can also determine their "delta status". This status tells you how the work item has changed, i.e., whether it's been added to the queue, deleted from the queue, or has been modified.

The way in which you retrieve changed work items depends on whether you are using pageable lists or single-block item access (i.e., the "**make**<type>**List**" and "**fetch**<type>**List**" methods), as described in the following subsections.

Note - There are two process attributes in the engine that specify the maximum number and percentage of the work item list that will be returned when requesting delta items. They are ***WIS_QCHANGE_MAX_CHANGES*** *and* ***WIS_QCHANGE_MAX_PCT****. By default, all delta items requested are returned, regardless the number or percentage. However, if one or both of these attributes is changed from the default, the server will return an* ***swDeltaTooBigErr*** *if the number or percentage exceeds the value specified in these attributes.*

Note - This section describes obtaining delta work items via the iProcess Server Objects interface. You can also obtain delta work items via a JMS topic publish/subscribe mechanism. For information, see *"Work Queue Deltas Via a JMS Topic" on page 112**. Note, however, that these two methods of obtaining work queue deltas work totally independent of one another (i.e., work queue deltas via a JMS topic does NOT make use of the process attributes mentioned above).*

Work Queue Deltas With Pageable Lists

The **refresh** method on **sPageableListR** contains an *aRefreshAction* parameter that specifies how the pageable list should be refreshed. This parameter can be used to specify that you want to refresh the list, and that you may want to retrieve only the changed (delta) work items from the pageable list. The following **SWPLRefreshType** enumeration value is passed in the *aRefreshAction* parameter to specify that you want to refresh the list "with delta":

- **swUpdateWithDelta** - This causes the **refresh** method to refresh the list "with delta", allowing you to then get the "delta count" and to retrieve the work items that have changed in the pageable list since the last refresh.

Note - The SWPLRefreshType enumeration contains other values that do not pertain to work queue deltas. For more information, see *"Refreshing a Pageable List of Work Items" on page 151*.

The **refresh** method returns a status indicator that tells you whether or not the new pageable list is different from the one you had. The pageable list status is enumerated by **SWPLStatusType**. The possible status values are:

- **swPLNoChange** - The work items in the pageable list have not changed.
- **swPLStatusOnly** - Only the status of the items in the pageable list have changed.
- **swPLChanged** - Items in the pageable list are different.
- **swPLOrphaned** - The pageable list is in a transition state between items being moved from one work queue to another.

If the return status is either **swPLStatusOnly** or **swPLChanged**, there are delta work items that can be retrieved.

Once you have refreshed the held pageable list of work items and there are delta items, you can use the following methods on the **sPageableListR** object to determine the number of changed work items, and to retrieve the changed work items:

- **getDeltaCnt** - This method returns the number of work items that have changed on the work queue since the last refresh. This number can be used to determine if you want to retrieve only the delta work items or the entire list. (This method returns -1 if a delta had not been requested on the previous refresh, or if you are using an older iProcess Objects Server that doesn't support deltas.)
- **getNextDeltaItem** - The first time this method is called after a refresh "with delta", the first delta work item (a **vWorkItem** object) is returned. The application can then loop on getNextDeltaItem for "delta count" iterations (obtained with **getDeltaCnt**) to get all changed work items. Alternatively, the application can loop on getNextDeltaItem with no count; the error **swNotFoundErr** will be returned when there are no more changed items.

  If the application calls this method when a delta was not requested on the refresh, the iProcess Objects Server will return an error indicating there is no delta (**swNoDeltaInfoErr**).

  One delta item is returned at a time so the application can start processing the changes as it receives them.

Note - If the pageable list contains vAWorkItem objects, each vWorkItem object will need to be cast to a vAWorkItem object.

Work Queue Deltas With Single-Block Item Access

The following methods are available on **sWorkQ** and **xWorkQ** to return the work items that have changed in the previously held work item list (the held work item list must have previously been created with the **makeWorkItemList** or **makeAWorkItemList** method):

- **fetchWorkItemListDelta** - This method refreshes the currently held list of work items (**vWorkItem** objects), and determines which work items have changed since the last refresh. A "state" object (**vWorkItemListState**) is returned by this method — the delta items are available on this state object.
- **fetchAWorkItemListDelta** - This method refreshes the currently held list of work items (**vAWorkItem** objects), and determines which work items have changed since the last refresh. A "state" object (**vAWorkItemListState**) is returned by this method — the delta items are available on this state object.

These methods contain an *aMaxCount* parameter that specifies the maximum number of changed work items (either vWorkItem or vAWorkItem objects) to return. If the number of changed work items exceeds the number specified in the *aMaxCount* parameter, a null array of work items is returned (in the **vWorkItemListState** or **vAWorkItemListState** object). If -1 is passed in the *aMaxCount* parameter, all changed work items are returned.

The **vWorkItemListState** and **vAWorkItemListState** objects returned by the fetchWorkItemListDelta and fetchAWorkItemListDelta methods contain a **getDeltaCnt** method that returns the number of changed work items in the list.

Delta Status

The **vWorkItem** (and derived **vAWorkItem**) objects that are returned by **getNextDeltaItem**, **fetchWorkItemListDelta**, or **fetchAWorkItemListDelta** contain a **getDeltaStatus** method that indicates how the work item has changed. The getDeltaStatus method returns one of the following (which are enumerated in **SWDeltaStatusType**):

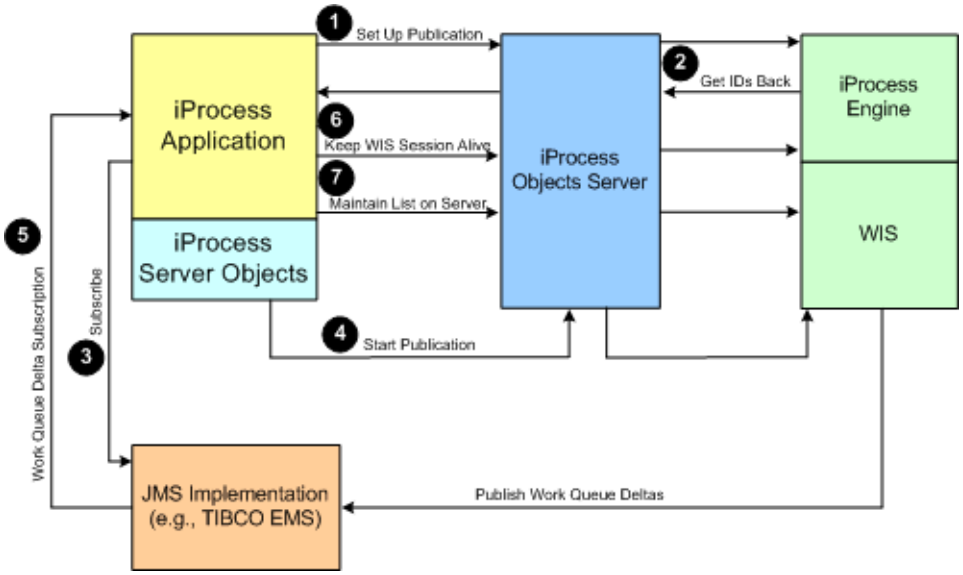
- **swNotDeltaItem** - This is returned by getDeltaStatus if the work item was not returned by one of the aforementioned methods (i.e., it is not a delta item).
- **swDeleted** - The work item was deleted from the work queue, i.e., it was released or forwarded to another work queue.
- **swAdded** - The work item is new to the work queue.
- **swModified** - The work item's status has changed, i.e., it has been locked, unlocked, or its deadline has expired.

Work Queue Deltas Via a JMS Topic

iProcess applications can also request that work queue deltas be obtained via a subscription to a topic in a JMS implementation (e.g., TIBCO Enterprise Message Service™ (EMS)).

Using this method, the application must first subscribe to a topic in JMS, then after getting the baseline list of work items in a work queue, all subsequent inserts, deletions, and updates to the work queue are published to the JMS topic by the WIS, which are then sent to the subscribing application.

The following diagram illustrates the general process of obtaining work queue deltas via a JMS topic.



*Note - Work queue deltas via a JMS topic works independently from the work queue delta functionality described in "Work Queue Deltas" on page 110 (i.e., obtaining work queue deltas via a JMS topic does NOT make use of the **WIS_QCHANGE_MAX_CHANGES** and **WIS_QCHANGE_MAX_PCT** iProcess Engine process attributes).*

The specific objects and methods you use to accomplish this depends on whether you are using pageable lists or single-block item access ("make" and "fetch" methods) in your application, as well as the interface from which you are programming. The following scenarios are described:

- JMS Deltas When Using Pageable Lists — JBase or RMI Interface
- JMS Deltas When Using Single-Block Item Access — JBase or RMI Interface
- JMS Deltas When Using Single-Block Item Access — XML Interface

Each of these scenarios is described in the following subsections.

JMS Deltas When Using Pageable Lists — JBase or RMI Interface

This section describes obtaining work queue deltas via a JMS topic when you are using pageable lists and programming to either the JBase or RMI interface. Reference the illustration on page 112.

1. The iProcess application sets up work queue delta publication by calling one of the following two method signatures on the **sWorkQ** object:

   - `sPageableListJ getAWorkItemListJMS(String aTopicName)`

   -
   ```
   sPageableListJ getAWorkItemListJMS(vWICriteria aWICriteria,
                                      int aItemsPerBlock,
                                      String aTopicName)
   ```

   The **getAWorkItemListJMS** method requires that you pass in a JMS topic name. This topic name will be used by the application when subscribing to the JMS topic, and by the WIS when publishing to the JMS topic.

   Note that the **getWorkItemsListJMS** method does not actually start the publication — it only tells the WIS that you are going to be retrieving deltas via a JMS topic and that you want it to generate IDs that will be used to coordinate the process.

   Note - There are also getAWorkItemListJMSHeld methods available to allow you to retrieve a "held" sPageableListJ. For information about held pageable lists, see "Held Pageable Lists" on page 155.

2. The **getAWorkItemListJMS** method returns an **sPageableListJ** object. This object contains two IDs that are returned by the WIS: **WorkQDeltaId** and **WorkQSyncId**. These IDs are needed by the application when calling the **startWorkQDeltaJMSPublish** method to start JMS delta publication from the WIS (see step 4). (The WorkQDeltaId is also needed for filtering the delta stream if multiple work queue deltas are on one topic.)

   Note that the primary purpose of the **getAWorkItemListJMS** method is to set up the JMS publication by the WIS. It is not to obtain the work item list — in fact, the WIS not does return a valid work item list when you call the **getAWorkItemListJMS** method. After starting JMS publication by calling the **startWorkQDeltaJMSPublish** method, you can then get the baseline list of work items off of the **sPageableListJ** object.

Also note that the counts on the **sPageableListJ** object are not valid at this point — after calling the **startWorkQDeltaJMSPublish** method, they become valid. The **sPageableListJ** object also does not contain a **refresh** method. Refreshing the list in this context does not make sense as the application is receiving any changes to the list via JMS messages.

3. The iProcess application must establish a subscription to the JMS topic using the topic name passed in the **getAWorkItemListJMS** method (see step 1).

   For information about how to do this, refer to the documentation for your JMS implementation (e.g., TIBCO EMS).

4. Start publication by calling the following method on the **sPageableListJ** object and passing the IDs that were returned by the WIS:

```
void startWorkQDeltaJMSPublish(String aWorkQDeltaId,
                               String aWorkQSyncId)
```

   The **startWorkQDeltaJMSPublish** method tells the WIS to:

   - establish the baseline list of work items, and
   - start publishing the work queue deltas to the JMS topic.

   Starting the publication process is a one-time event; you cannot start, then stop and start again. You can only start publication, then terminate it by destroying the work item list.

   After the **startWorkQDeltaJMSPublish** method is called, you can get the baseline list of work items from the **sPageableListJ** object.

5. The iProcess application must read the messages containing deltas as they are sent from JMS, then merge them into the baseline list. For information about the JMS messages, see "Work Queue Delta JMS Messages" on page 119.

6. Note that by default the WIS session will timeout after 8 hours of inactivity. This means that any iProcess application that has opened a queue for publication must periodically ensure some work is sent to the WIS by calling the following method on the **sSession** object:

```
void keepAliveWorkQDeltaJMSPublications()
```

   This tells the WIS that the application is still there and waiting to be informed of work queue deltas.

7. You must also do one of the following to prevent the iProcess Objects Server from destroying the work item list for which deltas are being published:

   - **Maintain the original reference to the work item list** - As long as you keep a reference to the work item list (that is, you don't set it to 'null', nor let it go out of scope), the list will never be garbage collected. This prevents the iProcess Objects Server from ever beginning to time out the list.

     As long as you keep a reference to the work item list, you will not have to do anything else to maintain the list on the server (besides calling **keepAliveWorkQDeltaJMSPublications** periodically -- see step above).

   - **Use the held ID to maintain a reference to the work item list** - If you cannot maintain a reference to the work item list, you can also use the held ID to reacquire the reference.

     Get the held ID:

```
String HeldId = WorkItemList.hold(true);
```

Save the held ID somewhere, then release the reference to the list. When garbage collection occurs, the iProcess Objects Server will start the time-out of the list. Before the list times out on the server, you must use the held ID to reacquire the reference to the list:

```
sPageableListJ WorkItemList = oWorkQ.getAWorkItemListJMSHeld(HeldId);
```

This restarts the time-out of the list on the server, so you need to continue to do this periodically, for as long as you need the JMS deltas to continue to be published.

*Note - If a **swSAL_FileIOErr** error is returned from one of the methods related to work queue deltas via a JMS topic, it is because the iProcess Engine is no longer publishing deltas to the JMS topic.*

For more information about the JMS delta methods and their parameters, see the iProcess Server Objects (Java) on-line help system.

JMS Deltas When Using Single-Block Item Access — JBase or RMI Interface

This section describes obtaining work queue deltas via a JMS topic when you are using single-block item access and you are programming to either the JBase or RMI interface. Reference the illustration on page 112.

1. The iProcess application sets up work queue delta publication by calling the following method on the **sWorkQ** object:

   ```
   vAWorkItemListState makeAWorkItemListJMS(vWICriteria aWICriteria,
                                            String aTopicName)
   ```

   The **makeAWorkItemListJMS** method requires that you pass in a JMS topic name. This topic name will be used by the application when subscribing to the JMS topic, and by the WIS when publishing to the JMS topic.

   Note that the **makeAWorkItemListJMS** method does not actually start the publication — it only tells the WIS that you are going to be retrieving deltas via a JMS topic and that you want it to generate IDs that will be used to coordinate the process.

2. The **makeAWorkItemListJMS** method returns a **vAWorkItemListState** object, which contains three IDs that are returned by the WIS: **HeldId, WorkQDeltaIdDeltaId** and **WorkQSyncId**. These IDs are needed by the application when calling the **startWorkQDeltaJMSPublish** method to start JMS delta publication from the WIS (see step 4). (The WorkQDeltaId is also needed for filtering the delta stream if multiple work queue deltas are on one topic.)

   Note that the primary purpose of the **makeAWorkItemListJMS** method is to set up the JMS publication by the WIS. It is not to obtain the work item list — in fact, the WIS does not return a valid work item list when you call the **makeAWorkItemListJMS** method. After starting JMS publication by calling the **startWorkQDeltaJMSPublish** method, you can then call the **fetchAWorkItemListJMS** method to get the valid baseline list of work items.

   Also note that the counts on the **vAWorkItemListState** object are not valid at this point — after calling the **startWorkQDeltaJMSPublish** method, they become valid.

3. The iProcess application must establish a subscription to the JMS topic using the topic name passed in the **makeAWorkItemListJMS** method (see step 1).

   For information about how to do this, refer to the documentation for your JMS implementation (e.g., TIBCO EMS).

4. Start publication by calling the following method on the **sWorkQ** object and passing the IDs that were returned by the WIS:

```
vAWorkItemListState startWorkQDeltaJMSPublish(String aHeldId,
                                              String aWorkQDeltaId,
                                              String aWorkQSyncId)
```

   The **startWorkQDeltaJMSPublish** method tells the WIS to:

   - establish the baseline list of work items, and
   - start publishing the work queue deltas to the JMS topic.

   Starting the publication process is a one-time event; you cannot start, then stop and start again. You can only start publication, then terminate it by destroying the work item list.

   After the **startWorkQDeltaJMSPublish** method is called, you can get the baseline list of work items by calling the following method on the **sWorkQ** object:

```
vAWorkItemListState fetchAWorkItemListJMS(String aHeldId,
                                          int aStartIndex,
                                          int aReturnCount)
```

   You may need to call this multiple times, incrementing the StartIndex each time by the number returned, if the ReturnCount isn't large enough to get all of the items in one call.

5. The iProcess application must read the messages containing deltas as they are sent from JMS, then merge them into the baseline list. For information about the JMS messages, see "Work Queue Delta JMS Messages" on page 119.

6. Note that by default the WIS session will time out after 8 hours of inactivity. This means that any iProcess application that has opened a queue for publication must periodically ensure some work is sent to the WIS by calling the following method on the **sSession** object:

```
void keepAliveWorkQDeltaJMSPublications()
```

   This tells the WIS that the application is still there and waiting to be informed of work queue deltas.

7. You must also call the **sWorkQ.fetchAWorkItemListJMS** method periodically to prevent the iProcess Objects Server from destroying the work item list.

   The iProcess Objects Server will destroy the work item list and stop publishing deltas to the JMS topic unless it receives a call on that list periodically. By default, the iProcess Objects Server will destroy the list after 15 minutes unless it receives a call on that list. You can prevent it from destroying the list by calling the **sWorkQ.fetchAWorkItemListJMS** method periodically. (Calling the **keepAliveWorkQDeltaJMSPublications** method prevents the WIS from timing out — it will not prevent the iProcess Objects Server from destroying the work item list.) The 15-minute default in which the iProcess Objects Server will destroy the list is configurable using the iProcess Objects Server **WQSAbandonedPeriod** configuration parameter.

   Also note that when you call the **sWorkQ.fetchAWorkItemListJMS** method, it will return the list. You probably won't need the list that is returned by that method — it is only for the purpose of maintaining the list on the server. Therefore, when you are calling **fetchAWorkItemListJMS** for the sole purpose of maintaining the list (i.e., not getting the baseline), you can pass -1 in *aReturnCount* to limit the amount of data returned.

*Note - If a **swSAL_FileIOErr** error is returned from one of the methods related to work queue deltas via a JMS topic, it is because the iProcess Engine is no longer publishing deltas to the JMS topic.*

For more information about the JMS delta methods and their parameters, see the iProcess Server Objects (Java) on-line help system.

JMS Deltas When Using Single-Block Item Access — XML Interface

This section describes obtaining work queue deltas via a JMS topic when you are using the XML interface. Reference the illustration on page 112.

1. The iProcess application sets up work queue delta publication by calling the following method on the **xWorkQ** object:

   ```
   vWorkQDeltaJMSId makeAWorkItemListJMS(String aId,
                                         vWICriteria aWICriteria,
                                         String aTopicName)
   ```

   The **makeAWorkItemListJMS** method requires that you pass in a JMS topic name. This topic name will be used by the application when subscribing to the JMS topic, and by the WIS when publishing to the JMS topic.

   Note that the **makeAWorkItemListJMS** method does not actually start the publication — it only tells the WIS that you are going to be retrieving deltas via a JMS topic and that you want it to generate IDs that will be used to coordinate the process.

2. A **vWorkQDeltaJMSId** object is returned by the **makeAWorkItemListJMS** method call. This object contains three IDs that are returned by the WIS: **HeldId, WorkQDeltaId** and **WorkQSyncId**. These IDs are needed by the application when calling the **startWorkQDeltaJMSPublish** method to start JMS delta publication from the WIS (see step 4). (The WorkQDeltaId is also needed for filtering the delta stream if multiple work queue deltas are on one topic.)

   Note that the primary purpose of the **makeAWorkItemListJMS** method is to set up the JMS publication by the WIS. It is not to obtain the work item list. To get the work item list, you must first start JMS publication by calling the **startWorkQDeltaJMSPublish** method, then you can call the **fetchAWorkItemListJMS** method to get the valid baseline list of work items (see step 4).

3. The iProcess application must establish a subscription to the JMS topic using the topic name passed in the **makeAWorkItemListJMS** method (see step 1).

   For information about how to do this, refer to the documentation for your JMS implementation (e.g., TIBCO EMS).

4. Start publication by calling the following method on the **xWorkQ** object and passing the IDs that were returned by the WIS:

   ```
   void startWorkQDeltaJMSPublish(String aId,
                                  String aHeldId,
                                  String aWorkQDeltaId,
                                  String aWorkQSyncId)
   ```

   The **startWorkQDeltaJMSPublish** method tells the WIS to:

   - establish the baseline list of work items, and
   - start publishing the work queue deltas to the JMS topic.

Starting the publication process is a one-time event; you cannot start, then stop and start again. You can only start publication, then terminate it by destroying the work item list.

After the **startWorkQDeltaJMSPublish** method is called, you can get the baseline list of work items by calling the following method on the **xWorkQ** object:

```
void fetchAWorkItemListJMS(String aId,
                           String aHeldId,
                           int aStartIndex,
                           int aReturnCount)
```

You may need to call this multiple times, incrementing the StartIndex each time by the number returned, if the ReturnCount isn't large enough to get all of the items in one call.

5. The iProcess application must read the messages containing deltas as they are sent from JMS, then merge them into the baseline list. For information about the JMS messages, see "Work Queue Delta JMS Messages" on page 119.

6. Note that by default the WIS session will timeout after 8 hours of inactivity. This means that any iProcess application that has opened a queue for publication must periodically ensure some work is sent to the WIS by calling the following method on the **xSession** object:

   ```
   void keepAliveWorkQDeltaJMSPublications()
   ```

   This tells the WIS that the application is still there and waiting to be informed of work queue deltas.

7. You must also call the **xWorkQ.fetchAWorkItemListJMS** method periodically to prevent the iProcess Objects Server from destroying the work item list.

   The iProcess Objects Server will destroy the work item list and stop publishing deltas to the JMS topic unless it receives a call on that list periodically. By default, the iProcess Objects Server will destroy the list after 15 minutes unless it receives a call on that list. You can prevent it from destroying the list by calling the **xWorkQ.fetchAWorkItemListJMS** method periodically. (Calling the **keepAliveWorkQDeltaJMSPublications** method prevents the WIS from timing out — it will not prevent the iProcess Objects Server from destroying the work item list.) The 15-minute default in which the iProcess Objects Server will destroy the list is configurable using the iProcess Objects Server **WQSAbandonedPeriod** configuration parameter.

   Also note that when you call the **xWorkQ.fetchAWorkItemListJMS** method, it will return the list in the XML results. You probably won't need the list that is returned by that method — it is only for the purpose of maintaining the list on the server. However, you will probably want to call **getXMLResults** and pass True in the *aClear* parameter, otherwise the XML array will continue to grow. (When you are calling **fetchAWorkItemListJMS** for the sole purpose of maintaining the list (i.e., not getting the baseline), you can pass -1 in *aReturnCount* to limit the amount of data returned.)

*Note - If a **swSAL_FileIOErr** error is returned from one of the methods related to work queue deltas via a JMS topic, it is because the iProcess Engine is no longer publishing deltas to the JMS topic.*

For more information about the JMS delta methods and their parameters, see the iProcess Server Objects (Java) on-line help system.

Work Queue Delta JMS Messages

The following is provided as an example of obtaining useful information from work queue delta JMS messages. Refer to the documentation for your specific JMS implementation provider.

- The following code sets up JMS subscription:

```
import javax.jms.*;

TopicConnectionFactory factory = new com.tibco.tibjms.
TibjmsTopicConnectionFactory(JMSserverUrl);

TopicConnection connection = factory.createTopicConnection(JMSuserName,
JMSpassword);

TopicSession topicSession = connection.createTopicSession(false,
javax.jms.Session.AUTO_ACKNOWLEDGE);

javax.jms.Topic topic = topicSession.createTopic(JMStopicName);

TopicSubscriber subscriber = topicSession.createSubscriber(topic);

connection.start();
```

- The following code gets the JMS message:

```
// 10 second timeout on receive
TextMessage message = (TextMessage)subscriber.receive(10000);
```

- This code prints the JMSDestination:

```
System.err.println("\nReceived message: JMSDestination = '" +
message.getJMSDestination() + "'");
```

  Example:

```
JMSDestination = 'Topic[topic.doug.1]'
```

- This code prints the WorkQID:

```
System.err.println("\nReceived message: Property.WQDID = '" +
message.getStringProperty("WQDID") + "'");
```

  Example:

```
Property.WQDID = '76F0923A-E65D-11DC-8557-001560ED88C8'
```

  The following table provides a list of available properties followed by example values:

Property Name	Example Value
WQDQueueName	swadmin@v11
IAPMessageType	WQD

Property Name	Example Value
IAPComputerName	corpServer
IAPProcedureName	CARPOOL
WQDID	76F0923A-E65D-11DC-8557-001560ED88C8
IAPNodeName	v11

- This code prints the message "Text" (the XML of the WQ Delta):

```
System.err.println("\nReceived message: Text = '" + message.getText() + "'");
```

Processing Work Items

Processing work items that are in a work queue involve the following concepts:

- **Locking** a work item gives you exclusive use of that work item so that you can modify it (it is also referred to as "opening" a work item).
- **Keeping** a work item causes changes that have been made to the locked work item to be saved. The work item is unlocked, then *kept* in the same work queue.
- **Releasing** a work item causes changes that have been made to the locked work item to be saved. The work item is unlocked, then *released*, which causes the work item to be removed from the work queue. The case advances to the next step in the procedure.

The following subsections describe the details of locking, keeping, and releasing work items.

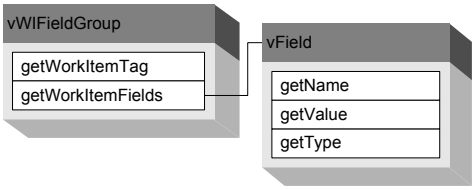
Locking Work Items

Locking a work item gives you exclusive use of that work item so that you can perform some action upon it. Note that "locking" a work item and "opening" a work item are synonymous.

The **lockItems** method on **sWorkQ** is used to lock one or more work items. This method requires an array of work item tags to identify the work items you want to lock.

You can also lock the first available work item in a list using the **lockFirstItem**, **lockFirstWorkItem**, and **lockFirstAWorkItem** methods. For more information, see "Locking the First Available Work Item in a List" on page 122.

For each work item that is locked, a **vWIFieldGroup** object is returned. The **vWIFieldGroup** object represents the set of fields on the work item. The **vWIFieldGroup** object contains a **getWorkItemFields** method, which allows you to access the individual fields for modification. It also has a **getWorkItemTag** method to allow you to identify the work item the fields are associated with.



Controlling Fields Returned when Locking Work Items

When you lock work items with the **lockItems**, **lockFirstItem**, **lockFirstWorkItem**, or **lockFirstAWorkItem** method, one or more **vWIFieldGroup** objects are returned, one for each work item that was locked. The **vWIFieldGroup** objects contain dependent **vField** objects, one for each field defined in that group of fields.

You can control which fields are returned from the server when a work item is locked by using the **vWIFGContent** object when calling **lockItems**, **lockFirstItem**, **lockFirstWorkItem**, or **lockFirstAWorkItem**.

The **vWIFGContent** object has the following parameters to control work item fields:

- *aWIFieldNames* - This parameter allows you to pass in an array of Strings identifying the work item fields you want returned with the work items.

  Note that the work item fields returned when locking the work items contain "Work Item Data." See "Case Data vs. Work Item Data" on page 166 for information about the difference between Work Item Data and Case Data.

- *aFieldsOption* - This parameter uses the **SWFieldsOptionType** enumeration to identify the option you have chosen for returning the work item fields upon locking a work item. The options are:
  - **ssoFormMarkings** - Return only visible markings on the form (based on conditional statements on the form).
  - **ssoAllMarkings** - Return all markings on the form (whether visible or not).
  - **ssoFieldList** - This isn't applicable when locking work items (to return a list of fields when locking work items, use the **vWIFGContent** object constructor with the *aWIFieldNames* parameter). This enumeration is returned by the **getFieldsOption** method if you view the content object after locking work items, and you passed in a fields list.

If the **vWIFGContent** object is constructed with no parameters, it defaults to returning all visible markings on the form.

Note - If you are using iProcess Modeler-produced forms, you can use any of the three constructors for **vWIFGContent** shown in the online-help system. However, if you are using a different type of form, you must use the constructor in which you pass in the array of field names; using the constructor with no parameters with a non-iProcess Modeler-produced form causes an empty list to be returned from the server. The constructor with the **SWFieldsOptionType** enumeration is only applicable to markings, which are only applicable to iProcess Modeler forms.

What's the Difference Between a "Lock" and a "Long Lock"?

There are two types of locks possible for a work item:

- A **lock** is one that is established from the **Work Queue Manager** (for those that are using the TIBCO iProcess Workspace (Windows)). If the work item is locked in this way, the **vWorkItem.isLocked** method will return True. This type of lock is not persistent — if the client exits, the work item is automatically unlocked.
- A **long lock** is one that is established using one of the following TIBCO iProcess Server Objects methods: **lockItems**, **lockFirstItem**, **lockFirstWorkItem**, and **lockFirstAWorkItem**. If the work item is locked using one of these methods, the **vWorkItem.isLongLocked** method will return True. This type of lock is persistent — if the client exits, the work item will remain locked.

The two types of locks are mutually exclusive — **isLocked** and **isLongLocked** cannot both be True. If a work item has been locked in the Work Queue Manager, and you attempt to lock it using TIBCO iProcess Server Objects, an error is returned telling you the work item is already locked. The same is true if it is already locked using TIBCO iProcess Server Objects, and you attempt to lock it in the Work Queue Manager.

Locking the First Available Work Item in a List

Methods are available that allow you to lock the first available work item in a list. (Note - To be able to use the methods described in this section, you must be using an iProcess Objects Server that has implemented MR 38404.)

The methods you use to lock the first available work item in a list depend on whether you are using pageable lists or single-block item access lists (for information about the difference in these list types, see "Working with Lists" on page 137), as follows:

Pageable Lists

The **sPageableListR** object provides the **lockFirstItem** method to lock the first available work item in the pageable list.

The lockFirstItem method contains a *aStartIndex* parameter, which allows you to specify that the search for an available (i.e., not currently locked) work item begin at the top of the pageable list, or at a specified index (zero based). When it finds the first available work item, the method locks the work item, then returns a **vWILocked** object, which provides methods that allow you to obtain the locked work item, associated field data, and the index at which the locked work item was located in the pageable list.

An *aWIFGContent* parameter is used to specify which work item fields are to be returned on the **vWILocked** object.

For more details about the **lockFirstItem** method, see the on-line help system.

Single-Block Item Access Lists

The **sWorkQ** and **xWorkQ** objects provide the following methods to lock the first available work item in a single-block item access list:

- **lockFirstWorkItem** - This method is used to lock the first available work item in a list of **vWorkItem** objects when you are using "single-block item access" lists (i.e., the **makeWorkItemList** method was used to create the list of **vWorkItem** objects).
- **lockFirstAWorkItem** - This method is used to lock the first available work item in a list of **vAWorkItem** objects when you are using "single-block item access" lists (i.e., the **makeAWorkItemList** method was used to create the list of **vAWorkItem** objects).

With both of these methods, the *aStartIndex* parameter allows you to specify that the search for an available (i.e., not currently locked) work item beginning at a specified index (zero based), or at the top of the list. When it finds the first available work item, the method locks the work item, then returns a **vWILocked** object, which provides methods that allow you to obtain the locked work item, associated field data, and the index at which the locked work item was located in the list.

An *aWIFGContent* parameter is used to specify which work item fields are to be returned on the **vWILocked** object.

For more details about the **lockFirstWorkItem** and **lockFirstAWorkItem** methods, see the on-line help system.

Unlocking a Work Item

A work item is automatically unlocked when it is “kept” or “released” (described later in this chapter).

The TIBCO iProcess Server Objects also provide the **sWorkQ.unlockItems** method to unlock one or more long-locked work items. This method causes all changes that have been made to the work items since it was locked to be discarded and unlocks the work items (the **isLongLocked** flag is set to False).

Any user can unlock a work item they have locked, but you must have system administrator authority (MENUNAME = ADMIN) to unlock a work item that another user has locked. That is the primary purpose of the **unlockItems** method — to allow a System Administrator to unlock another user’s work items. (See “User Attributes” on page 187 for information about the MENUNAME attribute.)

Discarding Changes made to a Locked Work Item

The **sWorkQ.undoItems** method is provided to discard changes that have been made to long-locked work items. This method causes all changes that have been made to the work item since the work item was locked to be discarded and unlocks the work item (the **isLongLocked** flag is set to False).

Has a Work Item been Locked/Opened?

When a work item is added to a work queue, its **isUnopened** flag is set to True, indicating that it has not been worked on yet. If you lock the work item (either through the Work Queue Manager or using TIBCO iProcess Server Objects), then keep it in the same work queue, its **isUnopened** flag is changed to False, indicating that it has been opened and worked on (but not necessarily modified).

The **vWorkQ** object also contains an **getUnopenedCnt** method, which returns the number of work items in the queue that have not been opened yet (i.e., the number that are new).

Determining who Locked a Work Item

You can determine the name of the user who currently has a work item locked by calling the **vWorkItem.getLockedBy** method.

Executing a Command when a Work Item is Locked

When a procedure is defined with TIBCO Business Studio, you can specify that a "command" be executed when the step (work item) is locked. If this has been defined in the procedure, the name of the command can be obtained with the **vCommand.getInitialExpr** method. The command may consist of any script or valid expression (for information about valid expressions, see the *TIBCO iProcess Expressions and Functions Reference Guide*).

Keeping Work Items

Keeping a work item causes changes that have been made to the work item to be saved. The work item is then *kept* in the same work queue.

The **keepItems** method on **sWorkQ** is used to keep one or more work items. This method requires an array of **vWIFieldGroup** objects to identify the work items to keep, as well as provide updated work item data to save. (The **lockItems** method returns these **vWIFieldGroup** objects when the work items are locked — see "Locking Work Items" on page 121. Note however, if you are locking a work item with the **lockFirstItem**, **lockFirstWorkItem**, or **lockFirstAWorkItem** method, you can get the **vWIFieldGroup** object from the **vWILocked** object that is returned from these lock methods.)

Note that the **vWIFieldGroup** objects returned from the lock methods cannot be modified. Instead, a new array of **vWIFieldGroup** objects must be created, identifying the fields that have been changed; this array is then passed to the **keepItems** method. This is to prevent the client from asking for lots of data, simply changing a single field, then returning all of the fields to the server. (It also supports the TIBCO iProcess Server Objects design that does not allow data to be altered on a Value Object (hence, no "set" methods) — all data must be passed in the Value Object constructors.)

When you keep a work item, data associated with that work item is written to "Work Item Data" (also known as "pack data"). This is an intermediate storage area for work items that are in work queues. See "Case Data vs. Work Item Data" on page 166 for more information.

Executing a Command when a Work Item is Kept

When a procedure is defined with TIBCO Business Studio you can specify that a "command" be executed when the step (work item) is kept. If this has been defined in the procedure, the name of the command can be obtained with the **vCommand.getKeepExpr** method. The command may consist of any script or valid expression (for information about valid expressions, see the *TIBCO iProcess Expressions and Functions Reference Guide*).

Releasing Work Items

Releasing a work item causes changes that have been made to the work item to be saved. The work item is then *released*, which causes the work item to be removed from work queue and the case to advance to the next step in the procedure.

The **releaseItems** method on **sWorkQ** is used to release one or more work items. This method requires an array of **vWIFieldGroup** objects to identify the work items to release, as well as provide updated work item data to save. (The **lockItems** method returns these **vWIFieldGroup** objects when the work items are locked — see "Locking Work Items" on page 121. Note however, if you are locking a work item with the **lockFirstItem**, **lockFirstWorkItem**, or **lockFirstAWorkItem** method, you can get the **vWIFieldGroup** object from the **vWILocked** object that is returned from these lock methods.)

Note that the **vWIFieldGroup** objects returned from the lock methods cannot be modified. Instead, a new array of **vWIFieldGroup** objects must be created, identifying the fields that have been changed; this array is then passed to the **releaseItems** method. This is to prevent the client from asking for lots of data, simply changing a single field, then returning all of the fields to the server. (It also supports the TIBCO iProcess Server Objects design that does not allow data to be altered on a Value Object (hence, no "set" methods) — all data must be passed in the Value Object constructors.)

When you release a work item, the "work item data" is written to "Case Data." This is the permanent storage area for data associated with the case. See "Case Data vs. Work Item Data" on page 166 for more information.

Validating Markings

The **releaseItems** method provides an *aValidateFields* parameter that allows you to ensure that all required markings are sent to the server upon release.

- If *aValidateFields* = **True**: This validates that the client application sends non-empty values for all required markings (swRequired) on the form. Note, however, if a required marking contains a non-empty value that came from a previous step, the client application does not have to send that value to the server. It also validates that display markings (swDisplay) are *not* sent to the server.
- If *aValidateFields* = **False** (the default), it bypasses the enforcement of marking types on the form.

Note - The *aValidateFields* parameter is only applicable if you are using iProcess Modeler-produced forms (it validates *markings*, which are only applicable on iProcess Modeler forms).

Executing a Command when a Work Item is Released

When a procedure is defined with TIBCO Business Studio you can specify that a “command” be executed when the step (work item) is released. If this has been defined in the procedure, the name of the command can be obtained with the **vCommand.getReleaseExpr** method. The command may consist of any script or valid expression (for information about valid expressions, see the *TIBCO iProcess Expressions and Functions Reference Guide*).

Automatically Releasing the Start Step

The **startCase** method provides an *aReleaseItem* parameter that allows you to specify that the start step be automatically released when the case is started. Setting this parameter to True causes the case to automatically proceed to the second step, resulting in the work item for the second step appearing in the work queue of the addressee of the second step. (Note that the *aReleaseItem* parameter is *only* relevant if the user starting the case is the addressee of the start step (or the addressee is defined as SW_STARTER).)

See “Keeping/Releasing the Start Step” on page 62 for more information.

What is an Orphaned Work Item?

When a work item is released, the **vWorkItem.isOrphaned** flag on that work item is set to True *on the local client copy.* The released work item will still appear in the work queue to prevent having to re-index the work items in the work queue after every release. The released work item will be removed from the local copy of the work queue when it is refreshed. The isOrphaned flag allows you to see which work items have been released without refreshing the work queue list.

Note that other clients cannot view the isOrphaned flag for work items that you release. That is, if another client had acquired a copy of the work queue (due to participation or if it's a group work queue) before you released the work item, the other client will not see the isOrphaned flag set to True after you release the work item. If they attempt to do any work item operation (lock, keep, release, etc.), they will get an "item not found" error. At this point, the other client needs to refresh their work queue to get the current work items (and their updated status).

Remember that a work queue is a "snapshot" of the work items in that queue at the time you acquire it. If other clients make changes to that work queue, those changes are not reflected in your view of the work queue until you refresh the queue.

Determining if a Work Item could not be Delivered to the Addressee

When a work item is released, the case advances to the next step in the procedure. The work item’s “addressee” specifies to whom the next work item in the case is to be sent. If the work item cannot be delivered to that addressee, the **vAWorkItem.isUndelivered** flag is set to True and the work item is placed in the **$undeliv** work queue (you must be logged in as the IPEADMIN user to access this work queue — for information about the IPEADMIN user, see “User Authority” on page 190).

Possible reasons for the work item not being able to be sent to the addressee is that the user or group specified as the addressee may no longer exist, or the addressee is specified as a field that evaluates to a user or group that does not exist.

Is the Work Item Directly Releasable?

A work item is considered “directly releasable” if there are no input fields on its form. Input fields include fields of type Required and Optional. Fields of type Display, Calculated, Hidden, and Embedded are not considered input fields.

If the work item is directly releasable, the **vWorkItem.isReleasable** flag is set to True.

Note that even though a work item is “directly releasable”, it must still be locked before it can be released.

Errors Resulting from Processing Work Items

When calling methods on the **sWorkQ** object to lock, unlock, keep, release, or undo multiple work items, the operation could possibly fail for one or more of the work items in the queue. Rather than totally backing out of the operation, a custom exception class is used to return two arrays, one for items that succeed and one for items that generated an error. See “Error Handling” on page 281 for information about handling errors.

Work Item Deadlines

When a step is defined using TIBCO Business Studio, a deadline may be specified on that step. If a deadline is defined, and the deadline expires, the process follows a “deadline link” to another step in the procedure, which is typically a notification to someone that the deadline has expired.

The actual definition of the deadline is part of the step definition. The **getDeadline** method on the applicable step object (vNormalStep, vSubProcCallStep, etc.) returns a **vDeadline** object, which contains deadline dates, times, criteria, etc.

If a work item has a deadline defined, the **vWorkItem.isDeadline** flag is set to True.

Once a work item is in a work queue, the following methods are available to provide information about the deadline on that work item:

- **vWorkItem.getDeadline** - Returns the date and time the deadline expires. This method will return the date and time “12/31/3000 11:15:00 PM” if a deadline has not been defined, or if a deadline has been defined but a condition in the deadline definition has not been satisfied.
- **vWorkItem.isDeadlineExp** - Returns a Boolean value that indicates whether or not the deadline has expired. This returns False if a deadline has not been specified for the work item.

The **vWorkQ** object also contains a **getFirstDeadline** method that returns the date and time of the earliest deadline in the work queue.

Deadline Withdrawal

Part of the deadline definition specifies whether or not the work item is withdrawn (removed) from the work queue when the deadline expires. (It’s common for the work item to remain in the work queue so that the required action on the work item can still be performed.) If the **Withdraw form from queue on expiry** box on the deadline definition dialog is checked when the deadline is created in TIBCO Business Studio, the work item represented by the step with the deadline is removed from the work queue if the deadline expires. In the example above, if this option is selected, when the deadline expires in the REVIEW step, the process flows to the REMIND step, and the work item for the REVIEW step is withdrawn from the work queue.

The deadline withdrawal option can be determined by calling the following method on the **vAWorkItem** object for the work item with the deadline:

- **isDeadlineAWD** - Returns True if the **Withdraw form from queue on expiry** box on the deadline definition dialog is checked.

Filtering and Sorting on Deadline Information

The following system fields are available for filtering and sorting work items on deadline information:

- **SW_HASDEADLINE** - Deadline set flag (1 - has deadline, 0 - all work items)
- **SW_DEADLINE** - Deadline date and time
- **SW_DEADLINEDATE** (filtering only) - Deadline date
- **SW_DEADLINETIME** (filtering only) - Deadline time
- **SW_EXPIRED** - Deadline expired flag (1 - has expired, 0 - all work items)

For information about using these system fields when filtering and sorting, see the appropriate *Filtering Work Items and Cases* chapter on page 208, page 234, or page 260 and "System Fields used in Sorting" on page 275.

Dynamically Recalculating Deadlines

A deadline (with or without a condition) is evaluated and/or calculated when a work item is sent to a work queue. It then remains in force until either the work item is released or the deadline expires.

In some situations, however, you may want to reset a deadline on a work item while it is still outstanding. For example, you may have used a deadline to set a review date for a customer's case in 6 month's time, but then for some reason want to bring that review date forward to 3 month's time.

You can force the iProcess Engine to re-calculate deadlines on all outstanding work items for a case by triggering an event on a particular step of the case using the **triggerEvent** method. The triggerEvent method must also update one or more field values used in setting the deadline — either in the expression that is used to calculate the deadline, or in an expression that is used to determine whether or not a deadline is set.

The **triggerEvent** method signature used to recalculate deadlines contains an *aDeadlineCalculate* parameter used to specify how deadlines are recalculated. The options, which are enumerated using **SWDeadlineCalculateType**, are as follows:

- **swNoReCalc** - Do not recalculate. This would be used when using **triggerEvent** to update outstanding work items, but you are not recalculating deadlines (the same method signature contains both the *aUpdateOutstanding* and *aDeadlineCalculate* parameters).
- **swCaseOnly -** Recalculate deadlines in the main case only.
- **swIncludeSubCases** - Recalculate deadlines in the main case and all sub-cases.

The way in which the iProcess Engine recalculates deadlines depends on whether or not a deadline condition has been defined, and whether it is an expression or period deadline, as follows:

Condition Type	Expression deadline is...	Period deadline is...
No condition.	Re-evaluated and set to the new value.	Not recalculated.
Condition now evaluates as true.	Re-evaluated and set to the new value.	Calculated as a period **from the original date that the work item was sent out**.
Condition now evaluates as false.	Removed.	Removed.

Note that you cannot use triggerEvent to recalculate a **Period** deadline that is not triggered by a condition, because no field values are involved in the deadline's calculation. The only way to force a re-calculation of such a deadline is to build logic into your procedure allowing you to withdraw the step and then resend it with the new deadline. However, if you do this, any changes made to the work item while it has been in the user's queue will be lost.

Keeping a Work Item that is Withdrawn

There are two ways in which you can specify that a work item be withdrawn from a work queue:

- When a deadline defined on the step expires. This is explained earlier in this chapter in "Deadline Withdrawal" on page 127.
- A "withdraw link" can be drawn when the step is defined in TIBCO Business Studio.

The step definition allows you to specify that under either of the conditions above, the work item should be kept in the queue instead of withdrawn. To specify this, on the Step Definition Status dialog in TIBCO Business Studio, check the **Don't delete work items on withdraw** box. You can determine whether or not this option has been checked by calling the **isKeepOnWithdrawal** method on the applicable step object (e.g., vNormalStep, vEventStep, etc.), or the **vWorkItem** object in a live case.

If the **Don't delete work items on withdraw** box is checked, and the work item would normally be withdrawn (because of a deadline expiration or release action of another step), the following occur instead:

- the work item is still considered "outstanding" (it is returned if you call **getOutstandingItems** on **sCaseManager**).
- the work item remains in the work queue (it is not "deleted").

When the work item is released (or sub-procedure case completes — if the step is a sub-procedure call), the following occurs:

- the normal release actions are NOT processed (as is normal for withdrawn work items).
- work item data in that work item is still written to case data upon release.

Participating in Another User's Work Queue

Allowing a user to "participate" in another user's or groups's work queue is called "participation." Participation allows the participant user to have access to the work items in the other user's work queue.

To be a participant of a work queue, a user who has been designated as an administrative supervisor of the work queue must create a "participation schedule" for the participant user. The participation schedule specifies the users and duration of the participation in the work queue.

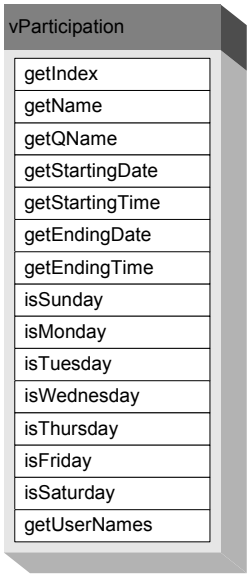
To define a participation schedule, a user must be designated as a supervisor of the work queue. See "Work Queue Supervisors" on page 134.

Note - Users with system administrator authority can access work queues of other users without being a "participant" user. This, however, comes at a cost. When a work queue is accessed without participation, a SAL session is automatically started for the user whose queue is being accessed, and that user is logged in internally. Note, however, that that user is not logged off automatically (the session

*will timeout automatically in the number of seconds specified in the **SALSessionTimeout** parameter). This overhead is not incurred if you use participation to give the user access to another user's work queue.*

Participation Schedules

A participation schedule is represented by the **vParticipation** object. This object contains the following participation elements:



- Names of the users who can participate in the work queue
- Dates to start and end the participation
- Times to start and end the participation each day
- Days of the week to allow participation

Each work queue can have multiple (or zero) participation schedules, and each participation schedule can have multiple users assigned to it who can participate during the period specified in the schedule.

You can determine the participation schedules that currently exist for a work queue using the following methods:

- **sWorkQManager.getParticipations** - This method sends a message to the server to retrieve an array of **vParticipation** objects, one for each participation schedule that has been defined for the specified work queue.
- **vAWorkQ.getParticipations** - This method returns an array of **vParticipation** objects, one for each participation schedule defined for the work queue represented by the local **vAWorkQ** Value Object.

Modifying Existing Participation Schedules

To make a change to an existing participation schedule, use the following method:

- **sWorkQManager.changeParticipation** - This method allows you to make a change to an existing participation schedule on the specified work queue. It requires that you construct a **vParticipation** object that specifies the new participation schedule, then pass it in the method call.

Creating Participation Schedules

To create a new participation schedule, you must construct a **vParticipation** object, then pass it as a parameter to the following method:

- **sWorkQManager.createParticipations** - This method sends the provided participation schedule to the server to take effect on the specified work queue.

Removing Participation Schedules

To remove/delete a participation schedule from a work queue, use the following method:

- **sWorkQManager.removeParticipations** - This method removes one or more participation schedules from the specified work queue. You must provide a **vParticipation** object identifying the participation schedule to remove.

Using the vDate and vTime Objects in Participation Schedules

The **vDate** and **vTime** objects are used in participation schedules to hold the starting and ending date/time. They can also be used to specify an "empty" date or time, which has various meanings, depending on the context.

Both **vDate** and **vTime** have two public constructors:

- One allows you to pass in a Date object to specify the starting or ending date/time.

```
public vDate(Date aDate)
public vTime(Date aTime)
```

- The other constructor, with no parameters, creates an "empty" **vDate** or **vTime** object:

```
public vDate()
public vTime()
```

If these objects are set to "empty," they take on the following meanings:

- *StartingDate* - Causes participation to begin on the next date allowed by the *IsSunday-IsSaturday* parameters.
- *EndingDate* - Causes participation to last indefinitely.
- *StartingTime* - Causes participation to start directly after midnight on the days that participation is allowed (according to the other parameters).
- *EndingTime* - Causes participation to end at midnight on the days that participation is allowed (according to the other parameters).

Forwarding/Redirecting Work Items to Another Work Queue

You can forward work items to another user's or group's work queue. Forwarding can be done in the following ways:

- **Manually Forwarding** - This allows you to forward one or more work items to a specified work queue.
- **Automatic Forwarding/Redirecting** - This allows you to set up a schedule so work items are automatically forwarded to another work queue during a specified time period. This is called "redirection."

These are described in detail in the following subsections.

Manually Forwarding Work Items

The **sWorkQ** object contains a **forwardItems** method that allows you to forward work items to a specified work queue. The following factors determine whether or not you can forward work items using **forwardItems**:

- **Work Queue Access** - To call the **forwardItems** method, you must have access to the work queue from which the work items are being forwarded.
- **The step's Forward Permission** - This is specified when the step is defined in TIBCO Business Studio. There is a **Forward** radio button on the step's **Step Status** dialog that specifies whether or not that step is forwardable (by default the step is forwardable). The step's forward permission is reflected in the **isForwardable** flag (which is available in the step definition,

vPermission, as well as the work item object, **vWorkItem**). The step's forward permission is used in conjunction with the user's forward permission (described below) to determine whether or not a user can manually forward a work item.

- **The User's Forward Permission** - The user's forward permission is specified in the USERFLAGS attribute. This permission specifies whether or not work items can be forwarded from this user's (or group's) work queue. Note that it is not the USERFLAGS attribute for the user that is calling **forwardItems** that is used. Internally, the system logs in as the user who owns the work queue — it's that user who must have the forwarding permission. You are really forwarding the work item on behalf of the owner of the work queue.

  The USERFLAGS attribute can have the following values/meanings:

  - " " - (Empty string) Work items from this user's work queue can be forwarded if the step's forward permission has been set. This is the default value. (This is called **Step Forward** in the User Manager.)
  - "**F**" - Any work item from this user's work queue can be forwarded, even if the step's forward permission has not been set. (This is called **Forward Any** in the User Manager.)
  - "**R**" - Work items from this user's work queue cannot be forwarded, even if the step's forward permission is set. (This is called **Forward None** in the User Manager.)

  See "User Attributes" on page 187 for information about modifying the value of the USERFLAGS attribute.

To manually forward a work item, call the **forwardItems** method and pass the tags for the work items you wanted forwarded, and a tag for the destination work queue.

Determining the Work Queues to which a Work Item can be Forwarded

You can determine the work queues to which a work item can be manually forwarded by using the following methods:

- **sWorkQ.getForwardToWorkQIds** - This method, returns an array of **vWorkQId** objects, one for each work queue to which the specified work item can be forwarded.
- **vAWorkItem.getForwardToWorkQIds** - This method, returns an array of **vWorkQId** objects, one for each work queue to which the work item represented by the local Value Object can be forwarded.

If you are calling the **getForwardToWorkQIds** method from **vAWorkItem**, you must use the following content object to specify that the "forward to work queue IDs" be returned from the server when the **vAWorkItem** objects are retrieved:

- **vAWIContent** - This content object can be created and passed as an input parameter with the **getAWorkItems** and **getAWorkItemList** methods. It specifies how much "content" (dependent objects) to return from the server with the work items. This content object inherits the **vWIContent** object.

  This content object has one method: **isWithForwardToWorkQIds**. This method allows you to determine how the forward to work queue ID flag was set when the content object was created.

Note that the list of "forward to work queue IDs" returned by **getForwardToWorkQIds** is not totally definitive; it provides a list of "available" work queues. Whether a work item can actually be forwarded to one of the work queues listed depends on the permissions described in the previous section, "Manually Forwarding Work Items".

Automatic Forwarding/Redirecting Work Items

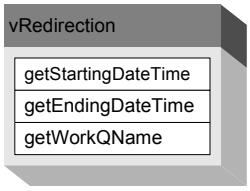
Automatically forwarding work items to another work queue is called “redirection.” Redirection allows you to “redirect” one user’s or group’s work items to the work queue of another user or group for a specified period of time.

To redirect work items, a user who has been designated as an administrative supervisor of the work queue must create a “redirection schedule.” The redirection schedule specifies to whom the work items are being redirected, as well as the date and time the redirection is to start and end.

To define a redirection schedule, a user must be designated as a supervisor of the work queue. See “Work Queue Supervisors” on page 134.

Redirection Schedules

A redirection schedule is represented by the **vRedirection** object. This object contains the following redirection elements:



- Name of the user or group to whom the work items are being redirected (**getWorkQName**)
- Date and time to start the redirection (**getStartingDateTime**)
- Date and time to end the redirection (**getEndingDateTime**)

When a work queue is created, a redirection schedule is automatically created with empty values (i.e., the work queue is not redirected to another user or group). There is only one redirection schedule per work queue.

You can access a work queue’s redirection schedule with the following methods:

- **sWorkQManager.getRedirection** - This method sends a message to the server to retrieve the work queue’s redirection schedule (**vRedirection** object).
- **vAWorkQ.getRedirection** - This method returns the redirection schedule (**vRedirection** object) for the work queue represented by the local **vAWorkQ** Value Object.

Modifying an Existing Redirection Schedule

Note that there is no method to “create” a redirection schedule since a schedule is automatically created when a work queue is created.

To modify a redirection schedule, use the following method:

- **sWorkQManager.changeRedirection** - This method allows you to change the redirection schedule for the specified work queue. It requires you to construct a **vRedirection** object, then pass it as a parameter in the method call.

Cancelling a Redirection Schedule

To cancel an existing redirection schedule, use the following method:

- **sWorkQManager.cancelRedirection** - This cancels the redirection, causing work items to no longer be redirected to another user’s work queue. Note that this does not remove the redirection schedule; it merely removes the values in the schedule.

Using the vDateTime Object in Redirection Schedules

The **vDateTime** object is used in redirection schedules to hold the starting and ending DateTime. It can also be used to specify an "empty" DateTime, which has various meanings, depending on the context.

The **vDateTime** object has two public constructors:

- One allows you to pass in a Date object to specify the starting or ending DateTime:

```
public vDateTime(Date aDateTime)
```

- The other constructor, with no parameters, creates an "empty" **vDateTime** object:

```
public vDateTime()
```

If this object is set to "empty," it takes on the following meanings:

- *StartingDateTime* - Causes the redirection to start immediately.
- *EndingDateTime* - Causes redirection to last indefinitely.

Work Queue Supervisors

To define a participation or redirection schedule, a user must be designated as a supervisor of the work queue. You can determine who the current supervisors are for a work queue using the following methods:

- **sWorkQueueManager.getSupervisorNames** - This method sends a message to the server to retrieve the names of the users who can supervise the specified work queue.
- **vAWorkQ.getSupervisorNames** - This method returns the names of the users who can supervise the work queue represented by the local **vAWorkQ** Value Object.

Note that the same list of user names returned by the **getSupervisorNames** method can be obtained from the QSUPERVISORS user attribute. See "User Attributes" on page 187 for information about QSUPERVISORS.

From the standpoint of a user, you can also determine which work queues the user has been authorized to supervise by calling the following method:

- **sUser.getSupervisedQIds** - This method returns an array of **vWorkQId** objects, one for each work queue for which the user is a supervisor.

Adding Work Queue Supervisors

To add a new supervisor to a work queue, use the following method:

- **sWorkQManager.addSupervisors** - This method allows you to specify that one or more users is a supervisor for one or more work queues. To add supervisors using this method, you must have system administrator authority (MENUNAME = ADMIN) — see "User Authority" on page 190 for information about the MENUNAME attribute.

Removing Work Queue Supervisors

To remove a supervisor from a work queue, use the following method:

- **sWorkQManager.removeSupervisors -** This method allows you to remove one or more users from the list of supervisors for the designated work queues. To remove supervisors using this method, you must have system administrator authority (MENUNAME = ADMIN) — see "User Authority" on page 190 for information about the MENUNAME attribute.

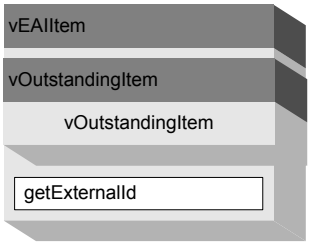
External Work Items

An external work item refers to a work item that is the result of a step that does not send a work item to a work queue, but rather passes the work item to an external third-party application. An example is an EAI step, which causes the TIBCO iProcess Engine to pass field data, the form definition, and a unique ID that identifies the external work item, to a third-party application. The third-party application uses the unique ID to pass the process flow back when it is finished.

The form definition that the engine passes to the third-party application is available by calling the **getExternalForm** method on the **sProcManager** object.

Note - The getExternalForm method was also previously available on the vEAIStep object.To provide better efficiency, it was removed from vEAIStep and added to the sProcManager object. This allows you to retrieve the external form data from the server only when it is needed. If you are using a TIBCO iProcess Objects Server with a message interface version older than 3.0.5, the getExternalForm method can still be used on vEAIStep. When using a TIBCO iProcess Objects Server with a message interface version of 3.0.5 or newer, this method should be called from sProcManager. If this method is called from vEAIStep when using a TIBCO iProcess Objects Server with a message interface version of 3.0.5 or newer, an empty String is returned.

Once the EAI step is outstanding, there is a **vEAIItem** object generated that represents the outstanding external work item. This object contains a **getExternalId** method, which returns the external ID generated by the engine that identifies the external work item. This external ID is sent to the third-party application when the EAI step becomes outstanding.



Releasing an External Work Item

The third-party application releases an outstanding EAI step using the **releaseEAIItem** method on the **sCaseManager** object. This method requires that the external ID that was passed to the third-party application be passed as a parameter in the releaseEAIItem method to identify the specific outstanding EAI step (there could be multiple EAI steps outstanding at one time).

When the releaseEAIItem method is called, you can optionally specify that the process flow proceed to a step that is different from the one defined to follow the EAI step in the procedure. This is done using the *aNextStep* parameter. If you specify an alternative next step with the *aNextStep* parameter, you can also use the *aDoActions* parameter to specify how the process flow should advance from the EAI step, as follows:

- If the *aDoActions* parameter is True, the actions defined for the work item being released are processed. This results in the process advancing to the next step as defined in the procedure, as well as the step specified in the *aNextStep* parameter.

- If the *aDoActions* parameter is False (the default), the process only advances to the step specified in the *aNextStep* parameter, but not the next step defined in the procedure.

You can also pass field data from the third-party application to be written to case data in the procedure.

9

Working with Lists

Introduction

There are two methods of accessing lists of items on the server:

- **Single-block Item Access -** This newer method of accessing lists allows you to access a specific block of items in a list. Each access requires you to specify the size of the block and the starting index. This method of accessing items is available in the following interfaces: JBase, RMI, SI, and XML. Note that if you are programmering to the SI or XML interfaces, you must use this method — the other method (pageable lists) is not available in those interfaces. For more information, see *Using Single-Block Item Access* on page 137.
- **Pageable Lists -** This older method of accessing lists is available in the following interfaces: JBase, RMI, EJB. For more information, see *Using Pageable Lists* on page 148.

Using Single-Block Item Access

This method of accessing lists of items on the server is a newer method than using pageable lists.

You must use this method to access lists of items if you are programming to the SI or XML interface. It is also available in the JBase and RMI interfaces (which are implementations of the SI interface), but not the EJB interface.

This method of accessing records on the server is accomplished by using the "**make**<*type*>**List**" and "**fetch**<*type*>**List**" methods:

- the **make**<*type*>**List** method creates a static list of the requested items on the server, and returns the requested number of items, starting at the specified index.
- the **fetch**<*type*>**List** method is used to get additional items from the static list created with the **make**<*type*>**List** method. This method is used only if you need to get additional items from the same list.

Note - This section is written from an object-oriented point of view. When the "make<type>List" and "fetch<type>List" methods are used in the JBase, RMI, or SI interfaces, data is returned in the form of objects; when they are used in the XML interface, data is returned in the form of an XML representation of those objects. If you are programming to the XML interface, and the text states "an object is returned", realize that for you the XML representation of the object is returned.

Making a List

The following **make**<*type*>**List** methods are available:

Server Object	"make" Method	Returns List of these Objects	List State Object
sNode / xNode	makeAGroupList	vAGroup	vAGroupListState
	makeOSUserList	vOSUser	vOSUserListState
	makeUserList	vUser	vUserListState
	makeWorkQIdList	vWorkQId	vWorkQIdListState
sWorkQ / xWorkQ	makeWorkItemList	vWorkItem	vWorkItemListState
	makeAWorkItemList	vAWorkItem	vAWorkItemListState
	makeAWorkItemListJMS[a]	n/a	vAWorkItemListState
sWorkQManager / xWorkQManager	makeWorkQIdList	vWorkQId	vWorkQIdListState
	makeWorkQList	vWorkQ	vWorkQListState
	makeAWorkQList	vAWorkQ	vAWorkQListState
sCaseManager / xCaseManager	makeACaseList	vACase	vACaseListState
	makePredictedItemList	vPredictedItem	vPredictedItemListState

a. The purpose of the makeAWorkItemListJMS method is to set up publication of work queue deltas to a JMS topic. For information, see *Work Queue Deltas Via a JMS Topic* on page 112.

Each of the **make**<*type*>**List** methods returns the object type indicated in the method name. For instance, **makeAWorkItem** returns a list of **vAWorkItem** objects.

make<type>List Method Input Parameters

At a minimum, all **make**<*type*>**List** methods require that you pass parameters specifying a start index, size of the block to return, and a "hold" flag. If called from the XML interface, a results ID is also required, as shown below.

From the XML Interface:

```
String makeWorkQIdList(String aId,
                       int aStartIndex,
                       int aReturnCount,
                       boolean aHold)
```

From the JBase, RMI, and SI Interfaces:

```
vWorkQIdListState makeWorkQIdList(int aStartIndex,
                                  int aReturnCount,
                                  boolean aHold)
```

where:

- *aId* specifies a results ID, which is a locator reference for the caller. Every call to an XML Server Object method takes a results ID as a parameter; that ID is reflected back in the **Id** attribute of the **<vResult>** tag in the XML data (see the example in the *Example XML Data* on page 297). (Note that if a NULL is passed in the *aId* parameter, the results ID defaults to the name of the method call (with an initial capital letter) — in this example, it would be "MakeWorkQ-IdList".)
- *aStartIndex* specifies the index number (zero based) of the first item to be returned from the static list.
- *aReturnCount* is the number of items you want returned from the static list, starting at the index number specified in *aStartIndex* (up to the number of items in the list — e.g., you may ask for 20 items, but only 15 exist).
- *aHold* is a flag that specifies whether or not to hold the static list after this method call. Pass True in this parameter if you are going to call the **fetch**<*type*>**List** method to get additional items off of the list. Pass False in this parameter if you are making a one-time request with the **make**<*type*>**List** method call. If False is passed in this parameter, the held ID returned by the **make**<*type*>**List** method is an empty string.

In addition to the parameters described above, some of the **make**<*type*>**List** methods also provide "criteria" and/or "content" parameters, as in the example below:

```
String makeWorkItemList(String aId,
                        vWICriteria aWICriteria,
                        vWIContent aWIContent,
                        int aStartIndex,
                        int aReturnCount
                        boolean aHold)
```

where:

- *aWICriteria* specifies filter and sort criteria for the list. (The specific criteria object passed in this parameter will depend on the type of objects in the list — this example is for work items, hence a "vWICriteria" object is passed.) For more information about filter and sort criteria, see *Filtering Work Items and Cases* on page 247.
- *aWIContent* specifies how much content (i.e., which dependent objects) to also return with the objects in the list. (The specific content object passed in this parameter will depend on the type of objects in the list — this example is for work items, hence a "vWIContent" object is passed.) For more information about content, see *Retrieving Dependent Objects on page 160*.

When filter and sort criteria is specified, the static list of objects created on the server will contain only the objects that satisfy the filter criteria, sorted in the order specified. The list state object returned by the **make**<*type*>**List** and **fetch**<*type*>**List** methods contains a count that tells you how many objects satisfied the filter criteria. (If you later call the **fetch**<*type*>**List** method, with *aRefresh*=True, the list is recreated using the same criteria and content information passed in on the original **make**<*type*>**List** method. The held ID remains the same if the list is refreshed.)

make<type>List Method Return Values

The values returned by the **make**<*type*>**List** methods depend on the interface from which the method is called, as follows:

From the XML Interface:

- **A Held ID (String) -** This is directly returned by the method call so that it can be used as an input parameter to the **fetch**<*type*>**List** method to identify the list that is held on the server. An empty string is returned for the held ID if False is passed in the *aHold* parameter (i.e., there is no need for a held ID if the list is not kept on the server).
- **"List State" Information -** This is returned internally in the form of a "list state" object, which provides count and other "state" information about the list. You must call the **getXMLResults** method to get the XML representation of the list state object. For information about the list state objects, see *List State Objects* on page 144).
- **The Block of Items Requested -** This is returned internally in the form of the object type requested. You must call the **getXMLResults** method to get the XML representation of the objects.

  Note that unlike the JBase, RMI, and SI interfaces, the block of items is not returned as part of the list state object. Instead, in the XML interface, the returned objects are a sibling to the list state object. (This is so that the returned items are located in the XML in the same location as items that are returned by method calls that are not associated with lists.)

From the JBase, RMI, and SI Interfaces:

- **A "List State" Object -** This object reflects counts and other "state" information about the list.

  Note that the held ID (which you will need if you want to call the **fetch**(*type*)**List** method) can be obtained from the list state object.

  The requested block of items is also contained in the list state object. Every list state object has a method call that returns the block of requested objects.

  For more information about list state objects see, *List State Objects* on page 144.

Fetching a List

The following **fetch**<*type*>**List** methods are available:

XML Server Object	"fetch" Method	Returns List of these Objects	List State Object
sNode / xNode	fetchAGroupList	vAGroup	vAGroupListState
	fetchOSUserList	vOSUser	vOSUserListState
	fetchUserList	vUser	vUserListState
	fetchWorkQIdList	vWorkQId	vWorkQIdListState
sWorkQ / xWorkQ	fetchWorkItemList	vWorkItem	vWorkItemListState
	fetchAWorkItemList	vAWorkItem	vAWorkItemListState
	fetchAWorkItemListJMS[a]	n/a	vAWorkItemListState
sWorkQManager / xWorkQManager	fetchWorkQIdList	vWorkQId	vWorkQIdListState
	fetchWorkQList	vWorkQ	vWorkQListState
	fetchAWorkQList	vAWorkQ	vAWorkQListState
sCaseManager / xCaseManager	fetchACaseList	vACase	vACaseListState
	fetchPredictedItemList	vPredictedItem	vPredictedItemListState

a. The purpose of the fetchAWorkItemListJMS method is to get the baseline list of work items after you've called startWorkQDeltaJMSPublish to tell the WIS to start publishing work queue deltas to a JMS topic. For information, see *Work Queue Deltas Via a JMS Topic* on page 112.

Each **fetch**<*type*>**List** method returns the object type indicated in the method name. For instance, **fetchAWorkItemList** returns a block of **vAWorkItem** objects.

fetch<type>List Method Input Parameters

At a minimum, all **fetch**<*type*>**List** methods require that you pass parameters specifying a held ID, a start index, size of the block to return, and a "hold" flag. If called from the XML interface, a results ID is also required, as shown below.

From the XML Interface:

```
void fetchWorkQIdList(String aId,
                      String aHeldId,
                      int aStartIndex,
                      int aReturnCount,
                      boolean aHold)
```

From the JBase, RMI, and SI Interfaces:

```
vWorkQIdListState fetchWorkQIdList(String aHeldId,
                                   int aStartIndex,
                                   int aReturnCount,
                                   boolean aHold)
```

where:

- *aId* specifies a results ID, which is a locator reference for the caller. Every call to an XML Server Object method takes a results ID as a parameter; that ID is reflected back in the **Id** attribute of the **<vResult>** tag in the XML data (see the example in the *Example XML Data* on page 297). (Note that if a NULL is passed in the *aId* parameter, the results ID defaults to the name of the method call (although, with an initial capital) — in this example, it would be "FetchWorkQIdList".)
- *aHeldId* identifies the list on the server. This ID was returned by the corresponding method call that created the list (e.g., makeWorkQIdList).
- *aStartIndex* specifies the index number (zero based) of the first item you want returned from the static list.
- *aReturnCount* is the number of items you want returned from the static list, starting at the index number specified in *aStartIndex* (up to the number of items in the list — e.g., you may ask for 20 items, but only 15 exist).
- *aHold* is a flag that specifies whether or not to hold the list on the server after this method call. Pass True in this parameter if you are going to fetch more items from the list; pass False if you are done with the list and will not be fetching any more items from it.

In addition to the parameters above, the **fetch**<*type*>**List** methods that return work items (**fetchAWorkItemList** and **fetchWorkItemList**) also contain an *aRefresh* parameter, for example:

```
void fetchWorkItemList(String aId,
                       String aHeldId,
                       int aStartIndex,
                       int aReturnCount,
                       boolean aHold,
                       boolean aRefresh)
```

where:

- *aRefresh* is a flag that specifies whether or not to *refresh* (i.e., rebuild) the list of work items, using the same criteria and content passed in the **make**<*type*>**List** method that was used to originally create the list, before returning the requested work items. This allows you to ensure that the list will contain work items that have been added to the work queue since the list was originally created, and will not contain work items that have been released from the work queue since the original list was created. The held ID remains the same if the list is refreshed.

  Also, the list state object returned by the **fetch**<*type*>**List** method contains a "list status" that tells you whether or not there was a change in the work item list after refreshing it (for more information, see *Work Item Status Information* on page 147).

fetch<type>List Method Return Values

The values returned by the **fetch**<*type*>**List** methods depend on the interface from which the method is called, as follows:

From the XML Interface:

- **"List State" Information** - This is returned internally in the form of "list state" objects, which provide count and other "state" information about the list. You must call the **getXMLResults** method to get the XML representation of the list state object. For information about the list state objects, see *List State Objects* on page 144).
- **The Block of Items Requested** - This is returned internally in the form of the object type requested. You must call the **getXMLResults** method to get the XML representation of the objects.

  Note that unlike the JBase, RMI, and SI interfaces, the block of items is not returned as part of the list state object. Instead, in the XML interface, the returned objects are a sibling to the list state object. (This is so that the returned items are located in the XML in the same location as items that are returned by method calls that are not associated with lists.)

From the JBase, RMI, and SI Interfaces:

- **A "List State" Object** - This object reflects counts and other "state" information about the list.

  The requested block of items is also contained in the list state object. Every list state object has a method call that returns the block of requested objects.

  For more information about list state objects see, *List State Objects* on page 144.

Fetch "If Changed" Methods

Two "fetch" methods are available that return work items only if there has been a change in work items on the list since you originally obtained it. The following methods are available on both the **sWorkQ** and **xWorkQ** objects:

- **fetchWorkItemListIfChanged**
- **fetchAWorkItemListIfChanged**

Both of these methods *always* refresh the list (there is no *Refresh* parameter — it is assumed to be True), and they return the requested work items only if *any* items in the *entire* list have changed (not just the specified range) since the list was originally created (with the **makeWorkItemList** or **makeAWorkItemList** method). If no work items in the list have changed since it was originally created, the returned list is zero length.

List State Objects

All **make**<*type*>**List** and **fetch**<*type*>**List** methods return *list state* information about the list.

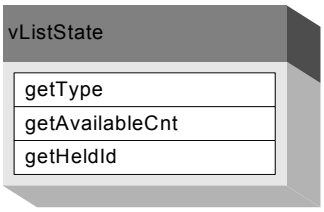
In the JBase, RMI, and SI interfaces, this is returned as an object; in the XML interface, this is returned in the form of the XML representation of the list state object.

The list state objects have the naming convention, **v**<*type*>**ListState**, where <*type*> is the type of object in the list. The following are the available list state objects:

- vACaseListState
- vAGroupListState
- vAWorkItemListState
- vAWorkQListState
- vOSUserListState
- vPredictedItemListState
- vUserListState
- vWorkItemListState
- vWorkQIdListState
- vWorkQListState

Base Object

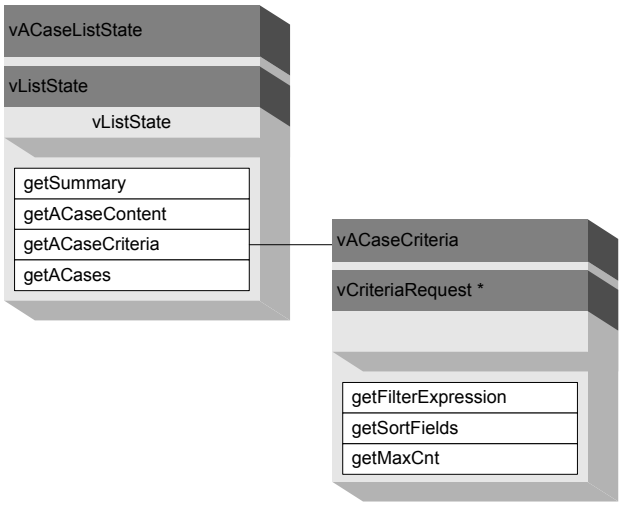
The **vListState** object is the base object from which all other list state objects are derived. This base object contains the following information:



- **Type** - Identifies the type of object in the list. This is enumerated using the **SWPageableListType** custom type. See the online help system for a list of the constants available in SWPageableListType.
- **Available Count** - This is the number of available objects in the static list on the server. If filter criteria was passed in the **make**<*type*>**List** method, only the objects that satisfied the filter criteria are included in this count.
- **Held ID** - This is used to identify the specific static list on the server. This held ID is returned by the **make**<*type*>**List** method, and is used as an input parameter to the **fetch**<*type*>**List** method to identify the list from which you want to fetch items. This will be an empty string if False was passed in the *aHold* parameter of the **make**<*type*>**List** method.

Filter and Sort Criteria Information

If the **make**<*type*>**List** or **fetch**<*type*>**List** method you call returns cases, work items, or predicted work items, the list state object will contain filter and sort criteria.



Filter and sort criteria is specified with the *"Criteria"* input parameter when you call the **make**<*type*>**List** method.

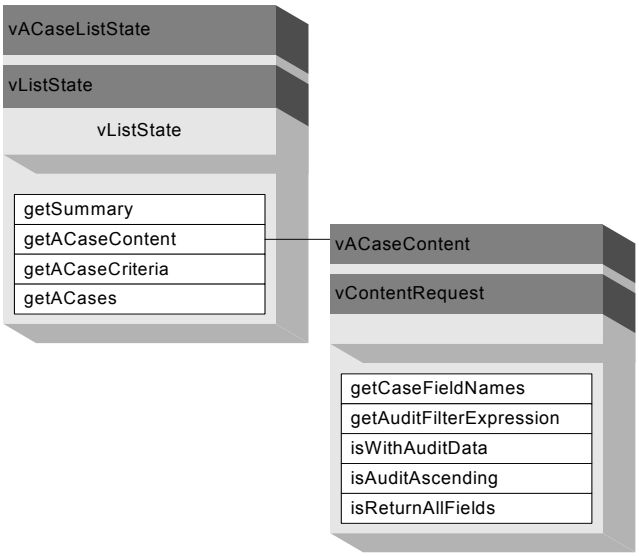
The following list state objects contain filter/sort criteria:

- **vACaseListState**
  - getACaseCriteria
- **vAWorkItemListState**
  - getWICriteria
- **vPredictedItemListState**
  - getPredictionCriteria
- **vWorkItemListState**
  - getWICriteria

The methods on these list state objects allow you to determine the filter/sort criteria that was specified (if any) when the list was created.

Content Information

Many of the **make**<*type*>**List** methods contain a *"Content"* input parameter that allows you to specify which dependent objects to also return with the requrested objects.



The list state objects returned by these methods also contain the content information.

The following list state objects contain content information:

- **vACaseListState**
  - getACaseContent
- **vAGroupListState**
  - getAGroupContent
- **vAWorkItemListState**
  - getAWIContent
- **vAWorkQListState**
  - getAWorkQContent
- **vUserListState**
  - getUserContent

- **vWorkItemListState**
  - getWIContent

The methods on these list state objects allow you to determine the content that was specified (if any) when the list was created.

Summary Information

If the **make**<*type*>**List** or **fetch**<*type*>**List** method you call returns cases, work items, or predicted work items, the list state object will contain summary information.

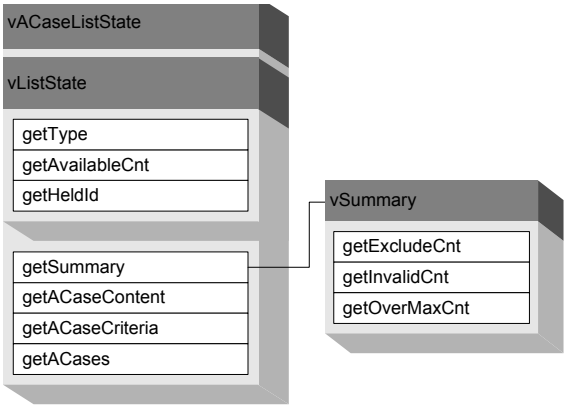
Summary information provides counts for the cases / work items in the list.

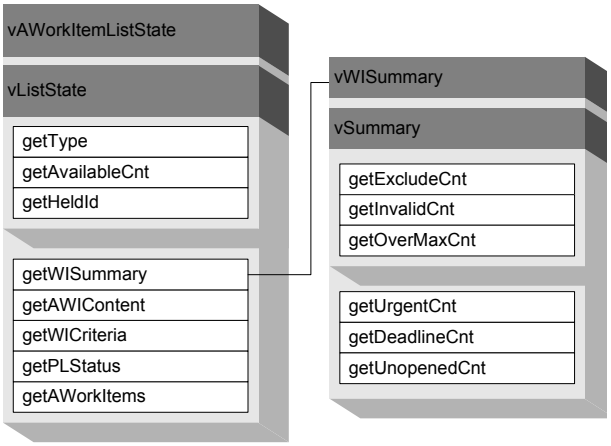
The following list state objects contain summary information:

- **vACaseListState**
  - getSummary
- **vAWorkItemListState**
  - getWISummary
- **vPredictedItemListState**
  - getSummary
- **vWorkItemListState**
  - getWISummary

All of the case / work item list state objects contain the **vSummary** object, which includes various counts about the objects on the list.

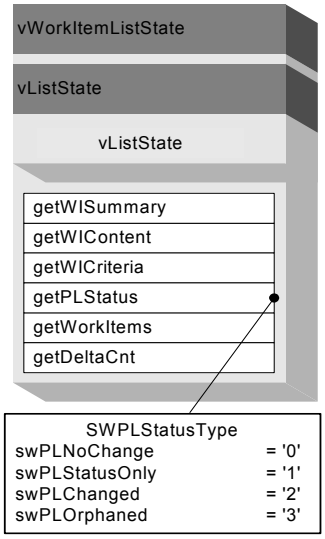
The list state objects for lists containing work items contain a **vWISummary** object, which includes additional counts for urgent, deadline, and unopened work items.





Work Item Status Information

The work item list state objects (**vAWorkItemListState** and **vWorkItemListState**) contain a **getPLStatus** method that allows you to determine if the work items in the list have changed since it was created.



Note that this status information is *only* applicable when you call the **fetchAWorkItemList** or **fetchWorkItemList** method, and pass in True for the *aRefresh* parameter. It is telling you, after the refresh, if the work items in the list have change (i.e., if there are any new work items or if any work items that were in the list are no longer there). (If you pass False in the *aRefresh* parameter when calling **fetchAWorkItemList** or **fetchWorkItemList**, a status of **swPLChanged** is always returned.)

The following list state objects contain status information:

- **vAWorkItemListState**
  - getPLStatus
- **vWorkItemListState**
  - getPLStatus

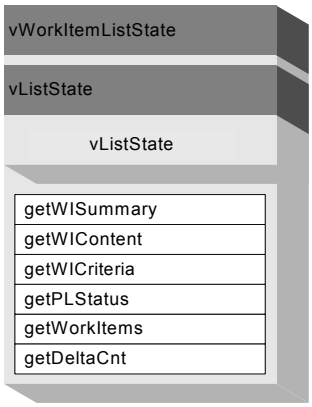
The **getPLStatus** method returns one of the following **SWPLStatusType** enumerations:

- **swPLNoChange** - There is no change in the work items.
- **swPLStatusOnly** - Only the status of one or more of the work items in the list has changed (i.e., some work items in the list have been locked and/or unlocked, but the same work items are in the list).
- **swPLChanged** - The work items in the list have changed.

Note - The "PL" in these names is for "Pageable Lists". The SWPLStatusType enumerations are also used with pageable lists.

Requested Items

If you are programming to the JBase, RMI, or SI interfaces, the requested items are returned in the list state object. For example, the **vWorkItemListState** object contains a **getWorkItems** method that returns an array of **vWorkItem** objects.



If you are programming to the XML interface, the requested items are returned as an XML representation of the objects when you call the **getXMLResults** method. For more information, see *XML Results* on page 296.

Using Pageable Lists

"Pageable lists" allow you to access lists, or "pages", of items from the server. This method of accessing lists is available in the following interfaces: JBase, RMI, and EJB. (It is *not* available in the SI nor the XML interfaces. You must use the "**make**<*type*>**List**" and "**fetch**<*type*>**List**" methods when programming to those interfaces — see *Using Single-Block Item Access* on page 137.)

Normally, Server Objects only act as a pass-through for data to and from the TIBCO iProcess Objects Server. The one exception is the "pageable list" object. (There are actually two pageable list objects, as shown in the illustration — **sPageableListR** derives from **sPageableList**.) These are special Server Objects that are used when dealing with potentially large lists of objects. They allow single-item access while maintaining continued access to the list of objects. The object's user can control the number of Value Objects that are created and held within a pageable list. This allows control over resource usage when dealing with a large number of objects.

Note - There is also an sPageableListJ object that is used specifically when requesting work queue deltas via a subscription to a topic in a JMS implementation. For information about using the sPageableListJ object, see *Work Queue Deltas Via a JMS Topic on page 112*.

All methods whose name begins with "get" and ends in "List" return a pageable list. Some examples are:

```
sPageableList getWorkQIdList()

sPageableListR getWorkItemList(vWICriteria aWICriteria,
                               vWIContent aWIContent,
                               int aItemsPerBlock)
```

The basic concept behind the pageable list is that instead of returning potentially thousands of objects (for example, vWorkItem objects) when a method is called, it instead returns "blocks" of objects as they are needed, significantly improving response time.

Pageable lists are used with the following types of items:

Item Type	Methods that return a pageable list	Value Object in Pageable List
Work items	sWorkQ.getWorkItemList	vWorkItem
	sWorkQ.getAWorkItemList	vAWorkItem
	sWorkQ.getAWorkItemListJMS[a]	vAWorkItem
Work queues	sWorkQManager.getWorkQIdList	vWorkQId
	sNode.getWorkQIdList	vWorkQId
	sWorkQManager.getWorkQList	vWorkQ
	sWorkQManager.getAWorkQList	vAWorkQ
Cases	sCaseManager.getACaseList	vACase
Groups	sNode.getAGroupList	vAGroup

Item Type	Methods that return a pageable list	Value Object in Pageable List
Users	sNode.getUserList	vUser
OSUsers	sNode.getOSUserList	vOSUser
Predicted Items	sCaseManager.getPredictedItemList	vPredictedItem

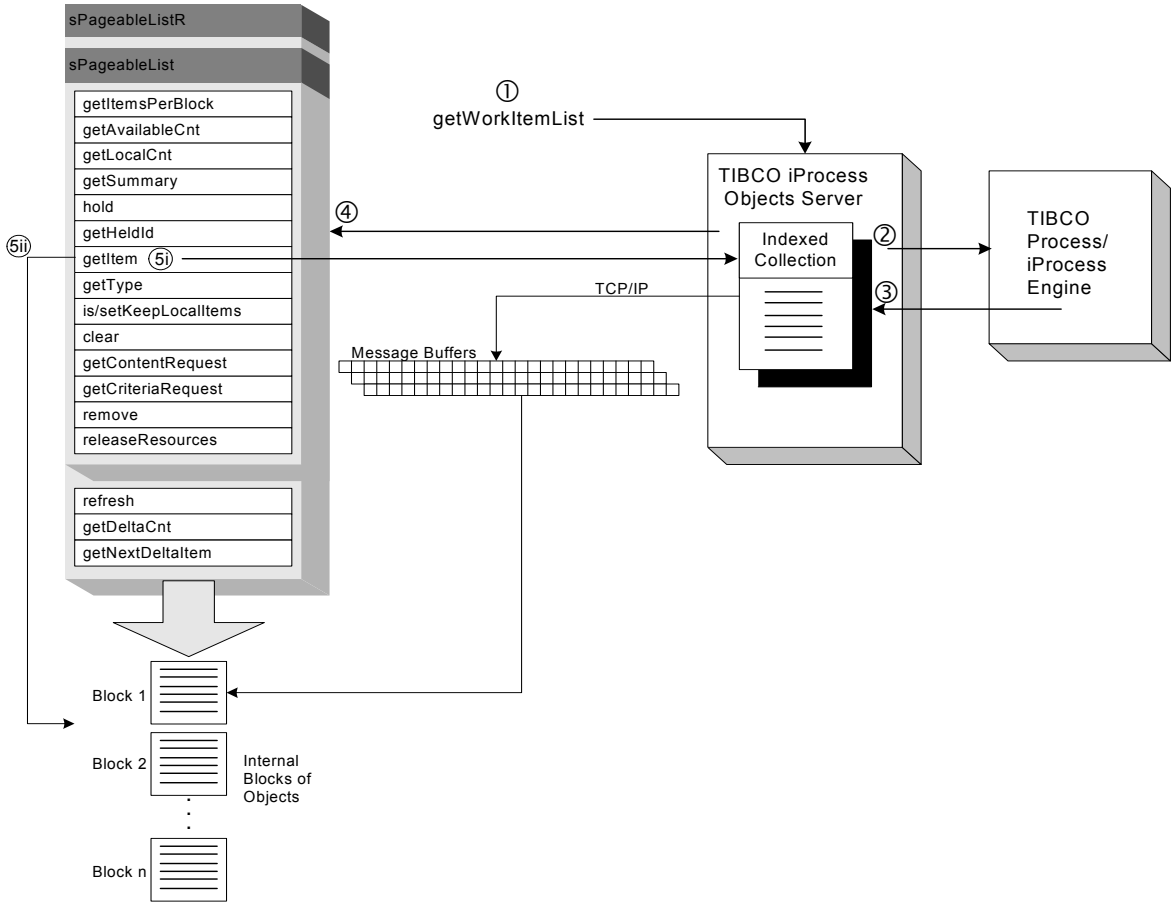
a. The purpose of the getAWorkItemListJMS method is to set up publication of work queue deltas to a JMS topic. For information, see *Work Queue Deltas Via a JMS Topic* on page 112.

These are the object types that are most likely to be in very large numbers on the TIBCO iProcess Objects Server, especially work items and cases.

Pageable lists containing work items work somewhat different than pageable lists containing other types of objects. The differences are described in the following sections.

Using Pageable Lists with Work Items

The following graphic illustrates how pageable lists are created when they contain work items or predicted work items:



1. When a method is called on a Server Object that returns a pageable list of work items, a message is sent to the TIBCO iProcess Objects Server requesting the work items.

2. The TIBCO iProcess Objects Server sends a message to the TIBCO Process/iProcess Engine (where the actual data is maintained) requesting the work item data.
3. The Work Item Server on the Process/iProcess Engine compiles an indexed collection of the requested work items and sends it to the TIBCO iProcess Objects Server.

   If filter, content, or sort criteria were passed in the method call on step 1, the work items in the indexed collection will only contain the items that satisfy the criteria, sorted in the order specified in the sort criteria.

   The indexed collection of work items is maintained on the TIBCO iProcess Objects Server until you either explicitly release it or it times out. See *Controlling Pageable List Resources* on page 157 for more information.

   At this point, no work items have been sent to the client — they are held in the indexed collection on the TIBCO iProcess Objects Server until you request a specific item.
4. An **sPageableListR** object is created and returned to the client. This object contains information about the indexed collection of work items on the TIBCO iProcess Objects Server, such as counts, status, and type.

   Note that the sPageableListR object acts somewhat like a Value Object in that it contains data (counts, status, etc.). It does not maintain its own session with the TIBCO iProcess Objects Server, but shares the user session of the original Server Object on which the method was called to create the sPageableListR.
5. From the **sPageableListR** object, you can call the **getItem** method to request a specific work item, providing the index number of the desired item.

   i. The **first time the getItem method is called**, a message is sent to the TIBCO iProcess Objects Server to request the work item. A “block” of work items containing the requested work item is sent to the sPageableListR, where the block is internally maintained. (The size of the block is controlled with the *aItemsPerBlock* parameter on the **get<ObjectClass>List** method you called to create the pageable list.) The requested work item is then returned to the client.

   ii. **All subsequent calls to getItem** cause it to first look in the internally held block(s) of work items to see if it already has the requested object. If it has it in its internal block(s), it returns that work item to the client. If the requested work item is not being held locally, a message is sent to the TIBCO iProcess Objects Server to retrieve the block containing the desired work item. The requested work item is then returned to the client.

   Note that the **isKeepLocalItems** flag on **sPageableList** controls whether or not more than one block will be held locally. If this flag is set to True, multiple blocks can be held locally. If it’s set to False, when another block is sent from the TIBCO iProcess Objects Server, the previous block is automatically removed, thereby minimizing the use of local memory. See *Client Resources* on page 158 for more information.

Refreshing a Pageable List of Work Items

You can “refresh” a pageable list containing work items, i.e., rebuild the pageable list so that it:

- no longer includes work items that have been released from the work queue since you created the pageable list,
- includes new work items that have been added to the work queue since you created your pageable list, and
- includes the current status of each of the work items in the work queue.

The **refresh** method causes the pageable list to be "refreshed" using the same filter and sort criteria as the original list, then report back on whether the work items in the list have changed. A message is sent to the server to request that the indexed collection of work items on the TIBCO iProcess Objects Server be updated with the most recent work item data from the TIBCO iProcess Engine, i.e., new work items in the queue are added to the collection, and released work items are removed. This also causes any internal blocks of objects in the **sPageableListR** on the client to be cleared, and the counts on the **sPageableListR** object to be updated.

The **refresh** method contains an *aRefreshAction* parameter that is used to specify how to refresh the list of work items. The values that can be passed in this parameter are enumerated using **SWPLRefreshType**. This possible values are:

- **swUpdate** - This causes the list to be updated (i.e., refreshed) if there has been a change in the list. (This is the default — there is a method signature that does not require a parameter.)
- **swRecreate** - This forces the list to be recreated, regardless of whether or not there has been a change in the list. Note that when this value is passed, the status returned by the method is always **swPLChanged** (for information about the return status, see the *Refresh Status* section below). This is applicable in situations when you are keeping or releasing work items from your own work queue, and there is no outside activity occurring (i.e., no new work items arriving). Because no outside activity has occurred, the status on the pageable list will indicate there’s been no change. If you attempt to refresh the list using **swUpdate**, it will not be recreated because the server thinks there hasn’t been any change — this value allows you to force the refresh in these situations.
- **swUpdateWithDelta** - This causes the list of work items to be refreshed “with delta”. This causes the server to make available to you all of the work items that have changed. This allows you to retrieve the work items that have been added to or deleted from the list, or those whose status has changed. For more information about retrieving delta work items, see *Work Queue Deltas* on page 110.

Refresh Status

The **refresh** method returns a status indicator that tells you whether or not the refreshed pageable list is different from the one you had. The refresh status is enumerated by **SWPLStatusType**. The possible status values are:

- **swPLNoChange** - The work items in the pageable list have not changed.
- **swPLStatusOnly** - Only the status of the items in the pageable list have changed (i.e., some work items in the list have been locked and/or unlocked, or deadlines have expired, but the same work items are still in the list).
- **swPLChanged** - Work items in the pageable list are different.
- **swPLOrphaned** - The pageable list is in a transition state between items being moved from one work queue to another.

*Note - There's an important distinction between calling **refresh** and calling the **get<ObjectClass>List** method again to get a new list of work items. If you call **refresh,** you are using the same user session and the same **sPageableListR** on the client. If you call the **get<ObjectClass>List** method again to re-acquire the pageable list of work items, a new user session is started and a new **sPageableListR** object is created — this is less efficient than calling **refresh**.*

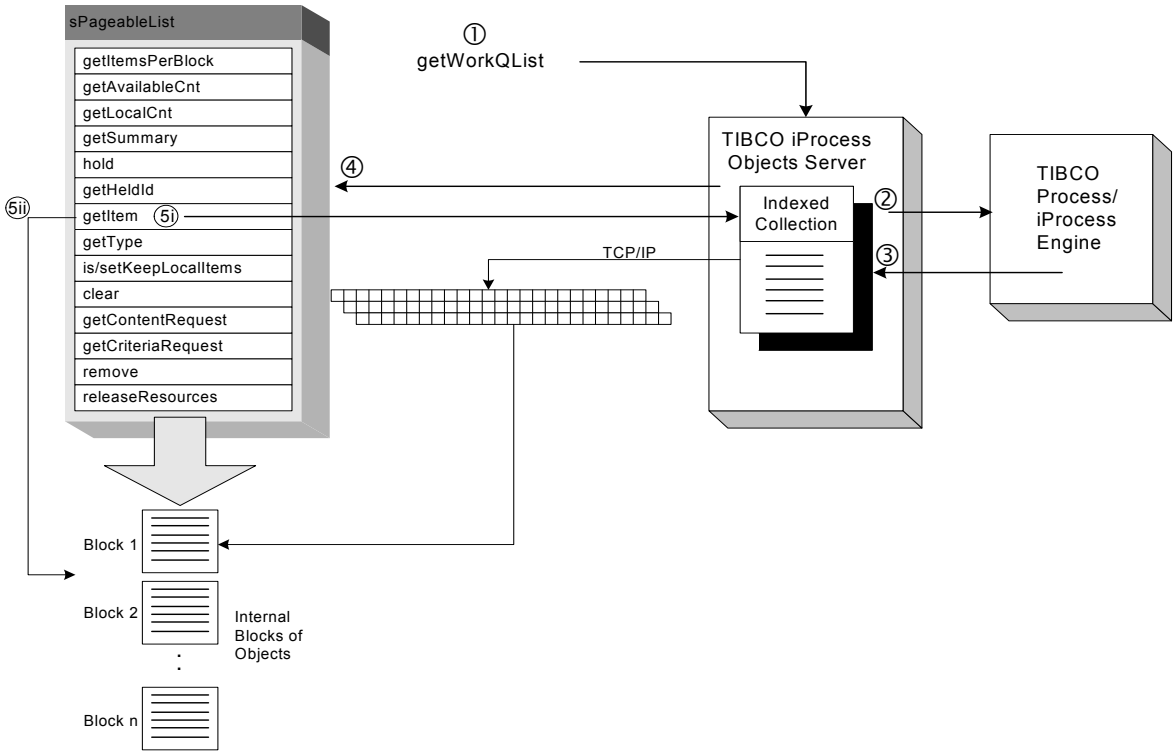
Using a Director or Multiple Instances of the TIBCO iProcess Objects Server

If you are using a TIBCO iProcess Objects Director to connect to a TIBCO iProcess Objects Server, or you are using multiple instances of the TIBCO iProcess Objects Server, be aware that a pageable list of work items or predicted work items is tied to a specific instance of the TIBCO iProcess Objects Server. If a pageable list of work items or predicted work items is created, that list can only be accessed on the specific instance of the TIBCO iProcess Objects Server where it was created. This is not just limited to getting the work items on the pageable list, but also to the method calls on work items obtained from the pageable list. This is because the list holds state to the Work Item Server.

Use the **getNodeId** method to get the **vNodeId** object for the particular instance of the TIBCO iProcess Objects Server to which you are connected so you can connect to that same instance at a later time.

Using Pageable Lists with Cases, WorkQs, Groups, Users, and OSUsers

The following graphic illustrates how pageable lists are created when they contain cases, work queues, groups, users, or OSUsers:



1. When a method is called on a Server Object that returns a pageable list containing cases, work queues, groups, users, or OSUsers, a message is sent to the TIBCO iProcess Objects Server requesting the objects.
2. The TIBCO iProcess Objects Server calls a function on the TIBCO Process/iProcess Engine (where the actual data is maintained) requesting the data.
3. The TIBCO Process/iProcess Engine compiles an indexed collection of the requested objects and sends it to the TIBCO iProcess Objects Server.

   If the method called is requesting a pageable list of *cases*, the method call may contain filter and sort criteria. If so, the cases in the indexed collection will only contain the cases that satisfy the criteria, sorted in the order specified in the sort criteria. (Filter and sort criteria don't apply to work queues, groups, users, and OSUsers.)

   The indexed collection of items is maintained on the TIBCO iProcess Objects Server until you either explicitly release it or it times out. See *Controlling Pageable List Resources* on page 157 for more information.

   At this point, no items have been sent to the client — they are held in the indexed collection on the TIBCO iProcess Objects Server until you request a specific item.

4. An **sPageableList** object is created and returned to the client. This object contains information about the indexed collection of items on the TIBCO iProcess Objects Server, such as counts, status, and type.

   Note that the sPageableList object acts somewhat like a Value Object in that it contains data (counts, status, etc.). It also does not maintain its own session with the TIBCO iProcess Objects Server — it shares the user session of the original Server Object on which the method was called that created the sPageableList.

5. From the **sPageableList** object, you can call the **getItem** method to request a specific case, work queue, group, user, or OSUser, providing the index number of the desired item.

   i. The **first time the getItem method is called**, a message is sent to the TIBCO iProcess Objects Server to request the item. ALL items in the indexed collection on the TIBCO iProcess Objects Server are sent to the message buffers. A "block" of items containing the requested object is created from the data in the message buffers, which is then sent to the sPageableList, where the block is internally maintained. (The size of the block is controlled with the *aItemsPerBlock* parameter on the **get<ObjectClass>List** method you called to create the pageable list.) The requested object is then returned to the client.

      Note that because *all* of the items in the indexed collection on the TIBCO iProcess Objects Server are sent to the message buffers the first time you access one of them, the first access may be slow if there is a large number of items in the indexed collection. All subsequent accesses will be very fast.

   ii. **All subsequent calls to getItem** cause it to first look in the internally held block(s) of objects to see if it already has the requested object. If it has it in its internal block(s), it returns that object to the client. If the requested object is not being held locally, another block of objects is created from the data in the messages buffers. The new block is sent to the sPageableList and the requested object is then returned to the client.

      Note that the **isKeepLocalItems** flag controls whether or not more than one block will be held locally. If this property is set to True, multiple blocks can be held locally. If it's set to False, when another block of objects is created from the message buffer data, the previous block is automatically removed. See *Client Resources* on page 158 for more information.

Held Pageable Lists

You can specify that a pageable list be “held”, which allows an application to disconnect from the TIBCO iProcess Objects Server, then reconnect at a later time and have access to the same collection of objects (same pageable list).

To hold a pageable list, call the **hold** method on **sPageableList** and pass True in the *aHoldList* parameter. This causes a “held ID” to be generated that identifies that specific pageable list. The held ID is returned by the **hold** method (it is also available by calling the **getHeldId** method on the **sPageableList** object, but the hold method must have been previously called). You are responsible for saving the held ID (typically as a cookie) so that it can be used at a later time as an input parameter to a method to retrieve the held pageable list. For example:

```
sPageableListR getWorkItemListHeld(String aHeldId)
```

There is a **get<ObjectClass>ListHeld** method for each of the object classes found on a pageable list. The “<ObjectClass>” portion of the method name tells you the class of object in the held pageable list, as follows:

Server Object	Held List Retrieval Method
sNode	getAGroupListHeld
	getOSUserListHeld
	getUserListHeld
	getWorkQIdListHeld
sWorkQManager	getWorkQListHeld
	getAWorkQListHeld
sWorkQ	getWorkItemListHeld
	getAWorkItemListHeld
	getAWorkItemListJMSHeld
sCaseManager	getACaseListHeld
	getPredictedItemListHeld

Note that held pageable lists containing work items and predicted work items are held by the TIBCO iProcess Objects Server, whereas held pageable lists containing other types of objects (cases, work queues, etc.) are actually held by the client. This means that if the TIBCO iProcess Objects Server should crash, you will lose any held pageable lists containing work items or predicted work items. And if the client application should crash, you will lose any held pageable lists containing the other classes of objects.

A held pageable list can only be used by one client connection at a time. Because of this, before you can retrieve a held pageable list, you must ensure that the previously referenced sPageableList object has been garbage collected. If garbage collection has not yet destroyed the sPageableList object, an **sw_NoPersistedIdErr** (201) will be thrown when you attempt to retrieve the held pageable list. For more information about this issue, see *Controlling Pageable List Resources* on page 157.

The isReturnAllFields Flag is always False on Held Pageable Lists

If you call **getACaseList**, **getAWorkItemList**, or **getWorkItemList**, and pass a **vACaseContent** or **vWIContent** object that has the *aIsReturnAllFields* parameter set to True, the pageable list that is returned will contain all available fields and show **isReturnAllFields**=True (which you can obtain from **sPageableList.getContentRequest**). The content and criteria objects are echoed directly back so that you can see the content/criteria that the pageable list was created with. However, if you "hold" this pageable list of cases or work items, then later retrieve the held list, the list that is returned will show **isReturnAllFields**=False (even though the list still contains the same fields from the original list). This is because when a list is held, the server resolves some values for efficiency reasons. In this case, "all fields" is resolved into a list of fields, so when the held pageable list is retrieved, it is not possible to determine if the original request was for a list of fields or a request for all fields.

Access Permissions

Only the user who creates a held pageable list (with a **get<ObjectClass>List** method) can re-access that list with a **get<ObjectClass>ListHeld** method, with the following exception:

- If the pageable list contains work items, and the list of work items is from a group work queue, any member of that group can re-access the held pageable list.

Pageable List Counts

There are a number of methods available that provide information about the number of items on a pageable list. They are:

On **sPageableList**:

- **Available Count (getAvailableCnt)** - This returns the total number of items available in the indexed collection on the TIBCO iProcess Objects Server. If the list contains work items or cases, and filter criteria was specified when the pageable list was created, this count will include only the work items or cases that satisfy the filter criteria.
- **Local Count (getLocalCnt)** - This returns the number of objects currently being held in the local bock(s) on the sPageableList. If **isKeepLocalItems** has been set to False, this count will always be less than or equal to the number of items per block (**getItemsPerBlock**).

On **vSummary**:

These counts are applicable only if the pageable list contains cases or work items.

- **Exclude Count (getExcludeCnt)** - This returns the number of cases or work items that did not satisfy the specified filter criteria, and therefore, were not included in the pageable list. (Note - This count may or may not be available, depending on which filtering enhancements have been implemented in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on page 193, page 221, or page 247.)
- **Invalid Count (getInvalidCnt)** - This returns the number of cases or work items that did not satisfy the filter criteria because the filter expression was invalid within the context of the item. For example, the filter expression references a field name that is not defined on all work items or cases. (Note - This count may or may not be available, depending on which filtering enhancements have been implemented in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on page 193, page 221, or page 247.)
- **Over Maximum Count (getOverMaxCnt)** - This returns the number of cases that were *not* returned from the server because the number returned was limited using the *aMaxCnt* parame-

ter on the **vACaseCriteria** constructor when retrieving a list of cases. This is applicable only when retreiving cases.

On **vWISummary**:
These counts are applicable only if the pageable list contains work items.

- **Urgent Count (getUrgentCnt) -** This returns the number of work items on the pageable list that are marked as urgent. A work item is marked as urgent if its priority value (vWorkItem.getPriorty) is less than or equal to a specific value. By default, the value is 10, although it can be modified in the **staffcfg** file.
- **Deadline Count (getDeadlineCnt) -** This returns the number of work items on the pageable list that have deadlines.
- **Unopened Count (getUnopenedCnt) -** This returns the number of work items on the pageable list that have not been opened (locked)

Controlling Pageable List Resources

Like all objects, sPageableList objects are freed by the JVM garbage collection. Garbage collection will occur some time after the pageable list variable is no longer referenced. Programmers, however, have no control over when garbage collection will occur. This is of particular importance regarding the retrieval of held pageable lists — if you attempt to retrieve a held pageable list, and garbage collection has not destroyed the original pageable list, it will result in an **sw_NoPersistedIdErr** (201) error. This is because there cannot be more than one client connection to a held pageable list, and you can't be assured of when garbage collection will destroy the original object.

This uncertainty of when garbage collection will occur also affects the freeing of resources on both the client and TIBCO iProcess Objects Server. The following provides some guidelines and recommendations to deal with these issues.

When the client is finished with a pageable list, it is strongly recommended that you do one of the following:

- call **sPageableList.releaseResources** - ***This is the preferred method***. This method releases the client-side pageable list resources without waiting for garbage collection to do it. If the pageable list is being held on the TIBCO iProcess Objects Server (it's a pageable list of work items or predicted work items), the list will still be available from the TIBCO iProcess Objects Server by a subsequent client connection.
- call **disconnect()** or **disconnect(False)** (i.e., with no parameter or with the *releaseAllResources* parameter set to False). This removes the sPageableList's usage of the TCP connection (socket) between the Server Object and the TIBCO iProcess Objects Server. The TCP connection will be closed when **disconnect** is explicity called on the Server Object that was used to create the **sPageableList** or in finalizer/garbage collection when the Server Object is destructed. If the pageable list is being held on the TIBCO iProcess Objects Server (it's a pageable list of work items or predicted work items), the list will still be available from the TIBCO iProcess Objects Server by a subsequent client connection.
- call **disconnect(True)** (i.e., with the *releaseAllResources* parameter set to True). This releases both client-side and server-side resources. This should be done only if you are completely done using a held pageable list. Future Server Objects will NOT be able to access held pageable lists that have been released using this method.

Note that held pageable lists are also automatically released using a timing mechanism. The timer starts when no client is referencing the pageable list (either **releaseResources** or **disconnect** has been called, or garbage collection has been run); the timer stops if a client retrieves the held list. The timer mechanism also depends on where the held pageable list is being held — held pageable lists containing work items or predicted work items are held by the TIBCO iProcess Objects Server; held pageable lists containing cases, work queues, groups, users, or OS users are held by the client:

- **Held by the TIBCO iProcess Objects Server -** Each pageable list that is held by the TIBCO iProcess Objects Server has a timer that defaults to 900 seconds (15 minutes). When this timer times out, the held pageable list is destructed.

  This timer is configured using the **WQSAbandonedPeriod** TIBCO iProcess Objects Server configuration parameter. For information, see the *TIBCO iProcess Objects Server Administrator's Guide.*
- **Held by the client -** Each pageable list that is held by the client has a timer that defaults to 900 seconds (15 minutes). When this timer times out, the held pageable list is destructed.

  If you would like to change this timer from the default value of 900 seconds, you must add a Registry key (Windows) or an environment variable (UNIX).

  The Windows Registry key is:

  ```
  HKEY_LOCAL_MACHINE\SOFTWARE\Staffware plc\Staffware SSO Client\PLHeldTimeOut
  ```

> *Note - If the software is installed on a 64-bit machine, the Registry path will include "Wow6432Node" as follows:*
>
> `HKEY_LOCAL_MACHINE\SOFTWARE\`**`Wow6432Node`**`\Staffware plc\...`

  The UNIX environment variable is:

  ```
  SSOClient_PLHeldTimeOut
  ```

  The timeout value is defined in seconds. The minimum it can be set to is 30 seconds; if set to a lower value, it will revert to 30 seconds. The maximum is 43,200 seconds (12 hours).

  Neither the Registry key nor the environment variable are automatically created for you. You must create them if you want to configure this value.

Client Resources

The internal blocks of objects maintained in the **sPageableList** object consume local memory as they are added to the pageable list. Several mechanisms are provided to allow you to control this memory usage:

- Setting the size of the blocks
- Automatically clearing blocks
- Explicitly clearing blocks

Setting the Size of the Blocks

The number of objects in each block that is retrieved from the TIBCO iProcess Objects Server (for work items) or created from the data in the message buffers (for cases, work queues, groups, users, and OSUsers) can be controlled by setting the *aItemsPerBlock* parameter on the **get<Object-Class>List** method to the desired number:

```
sPageableListR getWorkItemList(vWICriteria aWICriteria,
                               vWIContent aWIContent,
                               int aItemsPerBlock)
```

The default number of items per block is 20.

You can view the current number of items per block by calling:

```
sPageableList.getItemsPerBlock()
```

Automatically Clearing Blocks

As each new block of objects is added to the internally held blocks in the **sPageableList**, you can cause a previous block to be automatically removed by calling:

```
sPageableList.setIsKeepLocalItems(False)
```

This ensures that only one block at a time is kept internally on the **sPageableList** object. This property defaults to False.

Explicitly Clearing Blocks

When the objects in the internal blocks are no longer needed, you should free up local memory by calling:

```
sPageableList.clear
```

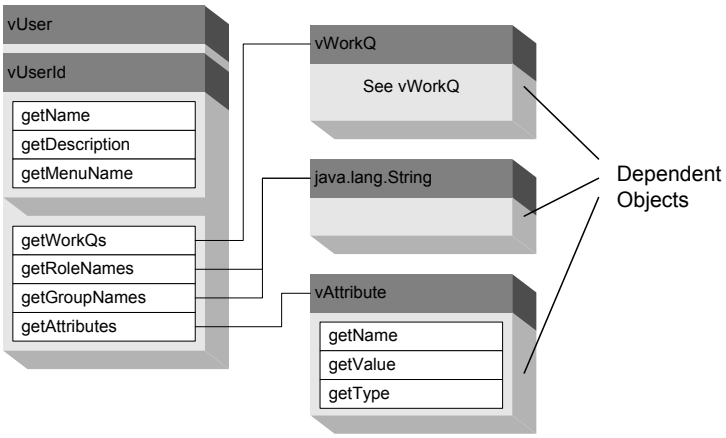
These local resources will also be automatically freed up when the pageable list is destroyed.

10

Retrieving Dependent Objects

Introduction

Many of the Value Objects that are retrieved from the TIBCO iProcess Objects Server with the **get<ObjectClass>** or **get<ObjectClass>List** methods have "dependent objects" that can also be returned with the primary object class. For example, the **vUser** object shown below has several dependent objects — **vWorkQ** objects, role and group names (both String objects), and **vAttribute** objects.



When you call the **getUsers** or **getUserList** method, you can include a "content" parameter (*aUserContent* in this particular case) that specifies how much content (i.e., dependent objects) to include with the **vUser** objects that are returned:

```
vUser[] getUsers(String[] aUserNames,
                 vUserContent aUserContent)


sPageableList getUserList(vUserContent aUserContent,
                          int aItemsPerBlock)
```

The content parameter requires you to pass a "content" Value Object (**vUserContent** in the examples above) that identifies how much content to return.

Content Request Objects

The following is a list of the available content request objects, as well as information about the content that can be filtered using the content request objects.

Content Request Object	Value Object Retrieved from Server	Dependent Objects
vACaseContent	vACase	• vAuditStep (audit data) • vField (case data)
vAGroupContent	vAGroup	• vAttribute (group attributes) • String (user names -- members of group)
vAProcContent	vAProc	• vProcSummary (dynamic counts for live cases of procedure)
vAWIContent	vAWorkItem	• vWorkQId (forwardable work queues)
vAWorkQContent	vAWorkQ	• vParticipation (participation schedules) • vRedirection (redirection schedule) • String (supervisor names) • vCDQPDef (CDQP definitions)
vGroupContent	vGroup	• vAttribute (group attributes)
vOutstandingItemContent	vOutstandingItem	• vNormalItem • vEventItem • vEAIItem • vSubProcCallItem • vDynamicSubProcItem • vGraftItem
vProcDefContent	vProcDef	• vAccessUserRef (users with admin authority) • vAccessUserRef (users with start authority) • String (nodes to which procedure is networked) • vStepId (steps in procedure) • vFieldDef (fields in procedure)
vStepContent	vStep	• vAction, vConditional, vDeadline, vAddressUserRef, vCommand, vPermission, vPriority (routing information)[a] • vFMarking (form marking data)
vUserContent	vUser	• vAttribute (user attributes) • String (group names -- to which user belongs) • String (role names -- to which user belongs) • vWorkQ (work queues -- to which user is assigned)

Content Request Object	Value Object Retrieved from Server	Dependent Objects
vWIContent	vWorkItem	• vField (Case Data - actually on the vCase object from which vWorkItem is derived) • vCDQP (CDQP data)
vWIFGContent	vWIFieldGroup	• vField (Work Item Data)

a. Note that all of the dependent objects listed do not apply to all step types.

Using Content Request Objects

Most of the content request objects contain Boolean “isWith” parameters that allow you to specify whether or not the dependent object represented by the “isWith” parameter should be included when retrieving the primary object class from the server.

For example, the **vGroupContent** object has an *isWithAttributes* parameter that specifies whether or not to include dependent **vAttribute** objects with the **vGroup** objects retrieved from the server. See the **vGroupContent** constructor below:

```
vGroupContent(boolean aIsWithAttributes)
```

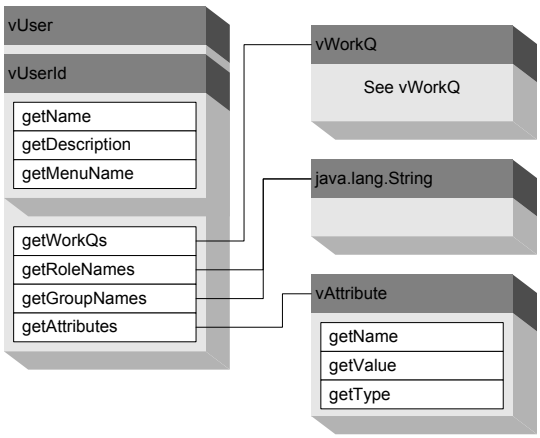
To include the dependent **vAttribute** objects, construct the **vGroupContent** object with *aIsWithAttributes* set to True, then use that object as an input parameter with the **sUser.getGroups** method.

Retrieving Dependent Objects

Once you have called a Server Object method that retrieves one or more Value Objects that contain dependent objects, a method is called on the Value Object to retrieve the desired dependent objects. The result of this method call indicates the presence of the dependent objects.

Using the **vUser** object as an example, calling **vUser.getAttributes**, results in one of the following being returned:

- An array of **vAttribute** objects. Obviously, the dependent **vAttribute** objects were asked for when retrieving the **vUser** objects.
- An empty array of **vAttribute** objects. The dependent **vAttribute** objects were asked for when retrieving the **vUser** objects, but there aren’t any.
- A NULL. This indicates that the dependent **vAttribute** objects were not asked for when the **vUser** objects were retrieved.



Controlling Case Data

When using the content request objects to control content on work items and cases, you can also specify which Case Data (i.e., the fields that contain the Case Data) to return with the work items and cases.

The constructors for the **vWIContent** and **vACaseContent** objects both have the following parameters, which are used to control Case Data returned with the work items and cases:

- *aCaseFieldNames* - This parameter allows you to pass in an array of Strings identifying the Case Data fields you want returned with the work items or cases.
- *aIsReturnAllFields* - This Boolean parameter allows you to specify that you want *all* Case Data fields returned with the work item or case (rather than having to list them all in the *aCaseFieldNames* parameter).

Use the *aIsReturnAllFields* parameter with caution, however, as it can result in a significant amount of data being sent across the network.

The **vWIContent** object also has similar parameters for specifying the Case Data Queue Parameter (CDQP) fields you want returned with the work items:

- *aCDQPNames* - This parameter allows you to pass in an array of Strings identifying the CDQP fields you want returned with the work items.
- *aIsReturnAllCDQPs* - This Boolean parameter allows you to specify that you want *all* CDQP fields returned with the work item (rather than having to list them all in the *aCDQPNames* parameter).

Controlling Fields Returned when Locking Work Items

When you lock one or more work items with the **lockItems** method, it returns an array of **vWIFieldGroup** objects, one for each work item that was locked.

When you lock the first available work item in a list with the **lockFirstItem**, **lockFirstWorkItem**, or **lockFirstAWorkItem** method, a **vWILocked** object is returned, which provides access to a **vWIFieldGroup** object for the work item that was locked.

The **vWIFieldGroup** objects returned by the lock methods contain dependent **vField** objects, one for each field defined in that group of fields.

You can control which fields are returned from the server when a work item is locked by using the **vWIFGContent** object when calling one of the available lock methods.

The **vWIFGContent** object has the following parameters to control work item fields:

- *aWIFieldNames* - This parameter allows you to pass in an array of Strings identifying the work item fields you want returned with the work items.

  Note that the work item fields returned when locking the work items contain “Work Item Data.” See *Case Data vs. Work Item Data* on page 166 for information about the difference between Work Item Data and Case Data.
- *aFieldsOption* - This parameter uses the **SWFieldsOptionType** enumeration to identify the option you have chosen for returning the work item fields upon locking a work item. The options are:
  - **ssoFormMarkings** - Return only visible markings on the form (based on conditional statements on the form).

- **ssoAllMarkings** - Return all markings on the form (whether visible or not).
- **ssoFieldList** - This isn't applicable when locking work items (to return a list of fields when locking work items, use the **vWIFGContent** object constructor with the *aWIFieldNames* parameter). This enumeration is returned by the **getFieldsOption** method if you view the content object after locking work items, and you passed in a fields list.

Note - If you are using iProcess Modeler-produced forms, you can use any of the three constructors for **vWIFGContent** shown in the on-line help system. However, if you are using a different type of form, you must use the constructor in which you pass in the array of field names; using the constructor with no parameters with a non-iProcess Modeler-produced form causes an empty list to be returned from the server. The constructor with the **SWFieldsOptionType** enumeration is only applicable to markings, which are only applicable to iProcess Modeler forms.

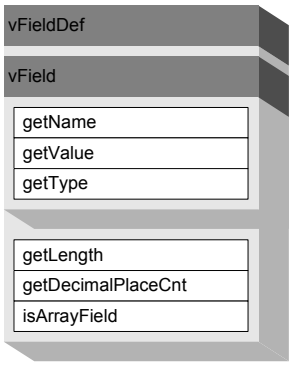
11

iProcess Fields

What is an iProcess Field?

An iProcess field represents a field that is defined in an iProcess procedure. Before a field can be placed on a form, the field must be defined in the procedure using TIBCO Business Studio.

iProcess fields are represented by the following two Value Objects:



- **vFieldDef** - This represents a field definition, as defined in TIBCO Business Studio. It includes the name and type of field (numeric, ASCII text, date, etc.), as well as the length and decimal place count definition. Since this object represents the *definition* of the field, it does not contain data — the **getValue** method will return the **SWEmptyField** object.

  The **getFieldDefs** method is used to access field definitions. This method, which is available on **sProcManager** and **vProcDef**, returns an array of **vFieldDef** objects:

  - If called from **sProcManager**, a message is sent to the server to return a **vFieldDef** object for every field definition for the specified procedure.
  - If called from **vProcDef**, one **vFieldDef** object is returned for each field that is defined in the procedure represented by the local Value Object.

- **vField** - This represents a field in the context of a live case. These Value Objects will contain a value if data has been entered into the field. There are two methods that can be used to obtain **vField** objects:

  - **getCaseFields** - This method on **vCase** returns an array of **vField** objects, one for each field in the live case represented by the case Value Object.

    The data in the fields returned by this method is considered "Case Data". See "Case Data vs. Work Item Data" on page 166 for more information.

  - **getWorkItemFields** - This method on **vWIFieldGroup** returns an array of **vField** objects, one for each field in the work item.

    The data in the fields returned by this method is considered "Work Item Data". See "Case Data vs. Work Item Data" on page 166 for more information.

Note that the **vField** objects that are returned by **getCaseFields** and **getWorkItemFields** may have been limited by content filtering when the **vCase** or **vWIFieldGroup** object was retrieved from the server. In other words, they may not include *all* fields in the case or work item; it depends on how many fields were requested when the **vCase** and **vWIFieldGroup** objects were retrieved from the server. For more information, see "Retrieving Field Data from the Server" on page 167.

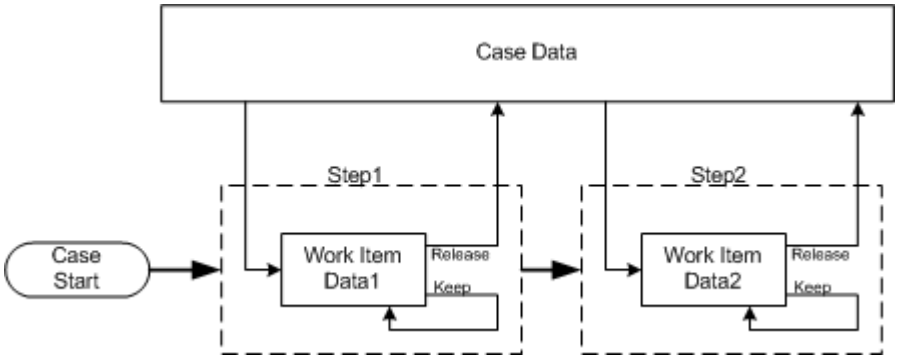
Case Data vs. Work Item Data

There are two types of data associated with a live case — **Case Data** and **Work Item Data**.

- **Case Data** is the "official" data for the case. This data is updated only when a work item is released. If there is another step in the procedure that results in another work item being added to another work queue, this data is copied to the next work item (where it becomes "Work Item Data" — see the illustration below).

  You can access Case Data by calling **getCaseFields**, then calling **getValue** on the desired **vField** object.
- **Work Item Data** is a copy of the Case Data that is taken when a work item is moved to a queue. This is a temporary holding area for the data associated with this work item that is maintained as long as the work item is kept in the work queue. When the work item is released, the data is written to Case Data. (Work item data is sometimes called "pack data" or "packfile" data.)

  You can access Work Item Data by calling **getWorkItemFields**, then calling **getValue** on the desired field. (Note that you must first lock the work item (using the **lockItems**, **lockFirstItem**, **lockFirstWorkItem**, or **lockFirstAWorkItem** method), which returns a **vWIFieldGroup** object, from which **getWorkItemFields** can be called. In other words, Work Item Data is only accessible from locked work items.)



Including Field Data when Starting a Case

If you start a case "with field data" (i.e., you are passing field names in the *aFields* parameter on the **startCase** method), you also have the option of "keeping" or "releasing" the step when the case is started (using the *aReleaseItem* parameter on the **startCase** method). The effect this has on Case Data and Work Item Data is described below:

- If you "keep" the start step when the case is started (the **startCase** *aReleaseItem* parameter = False), the data that is passed with **startCase** is copied to the Work Item Data of the first step. The Case Data will remain empty until the first step is released.
- If you "release" the start step when the case is started (the **startCase** *aReleaseItem* parameter = True), the data that is passed with **startCase** is copied to the Case Data and the flow moves to the next step. This is equivalent to doing a "keep," then a "release" on the first step.

Note - If a field has been defined as a Numeric field on the form, and you are passing a value into that field when starting a case, you MUST pass in a Double, otherwise an exception is thrown.

Setting Case Data

You can modify the Case Data for one or more fields in the case using the **setCaseData** method on **sCaseManager**. This method requires you to pass in an array of **vField** objects that contain the names of the fields and the values you want assigned to those fields:

```
void setCaseData(String aCaseTag,
                 vField[] aFields)
```

Note - You must be using a TIBCO iProcess Engine to use the SetCaseData method.

Uninitialized Fields

A field can have a value of **SW_NA**. This means the value has never been set, therefore, it is uninitialized. It would be the equivalent of specifying a variable in Java but not assigning a value to it. In Java, the variable would have a NULL value (an uninitialized variable).

The iProcess Server Objects interface supports this concept of uninitialized field through the special **SWEmptyField** object. The **vField.getValue** method returns an object, therefore if the field value is SW_NA, the **vField.getValue** method will return the **SWEmptyField** object.

Assigning an SWEmptyField object to a field resets the field to the uninitialized state.

Parallel Steps

Because each step has its own Work Item Data, this can create problems if your procedure has parallel steps that change a common field. As each step is released, it copies its changed Work Item Data to the Case Data, overwriting the data that was written to Case Data by any previous parallel steps. Any other parallel steps that have not been released yet, do not see the new Case Data — they still only see the Work Item Data that belongs to that step. (Note that to overwrite any previous case data, the field must actually be sent to the server when you release the work item with the **releaseItems** method. For more information, see "Passing Field Data when Keeping/Releasing Work Items" on page 169.)

In the end, the value of the common changed field in the Case Data will be the value from the last parallel step to be released.

Retrieving Field Data from the Server

To provide more control over resource usage, field data (i.e., **vField** objects) is not returned from the TIBCO iProcess Objects Server unless it is specifically requested. This minimizes network traffic and provides a faster response.

You must specify which fields you want returned from the server when you:

- retrieve cases
- retrieve work items
- lock work items

This is accomplished with the **vACaseContent**, **vWIContent**, and **vWIFGContent** objects, respectively. These "content" objects are used as input parameters on methods used to retrieve cases and work items, and lock work items.

Including Field Data when Retrieving Cases

To include fields when retrieving cases from the server, do the following:

1. Construct a **vACaseContent** object, specifying the desired fields in the *aCaseFieldNames* parameter, or pass True for the *aIsReturnAllFields* parameter to request all fields to be returned.
2. Pass the **vACaseContent** object as an input parameter with either the **getACases** or **getACaseList** method. These methods return **vACase** objects.
3. Use the **getCaseFields** method to get the **vField** objects that were returned from the server.

Including Field Data when Retrieving Work Items

Remember that when you retrieve a work item from the server, you also receive the **vCase** base class. It's this base class from which fields on the work item are available. Note, however, that even though the fields are from a *work item*, they contain *Case Data.* The *Work Item Data* is not available until you lock the work item — see the next subsection.

To include fields when retrieving work items from the server, do the following:

1. Construct a **vWIContent** object, specifying the desired fields in the *aCaseFieldNames* parameter, or pass True for the *aIsReturnAllFields* parameter to request all fields to be returned.
2. Pass the **vWIContent** object as an input parameter with either the **getWorkItems**, **getWorkItemList**, or **getAWorkItemList** method. These methods return either **vWorkItem** or **vAWorkItem** objects.
3. Use the **getCaseFields** method to get the **vField** objects that were returned from the server. (The getCaseFields method is on **vCase** — the work item objects are derived from vCase.)

Including Field Data when Locking Work Items

Locking work items results in an array of **vWIFieldGroup** objects being returned, each representing a group of fields on a work item. When you lock work items, you specify which fields are to be returned, and therefore, represented by the **vWIFieldGroup** object.

To specify which fields are to be returned from the server when locking work items, do the following:

1. Construct a **vWIFGContent** object, specifying the desired fields in the *aWIFieldNames* parameter. You can also use the *aFieldsOption* parameter to specify that you want, for example, only visible fields returned, or all fields returned, etc. See the **SWFieldsOptionType** enumeration in the on-line help for more information about the options available.
2. Pass the **vWIFGContent** object as an input parameter with the **lockItems**, **lockFirstItem**, **lockFirstWorkItem**, or **lockFirstAWorkItem** method.

   If you are using the **lockItems** method, an array of **vWIFieldGroup** objects are returned, one for each work item that was locked.

   If you are using the **lockFirstItem**, **lockFirstWorkItem**, or **lockFirstAWorkItem** method, a **vWILocked** object is returned, from which you can obtain a **vWIFieldGroup** object for the work item that was locked.
3. Use the **getWorkItemFields** method on the **vWIFieldGroup** object to get the **vField** objects that were returned from the server. These **vField** objects contain *Work Item Data.*

Note - For more information about content request objects, see *"Retrieving Dependent Objects" on page 160.*

Case Data Queue Parameter Fields

Case Data Queue Parameter (CDQP) fields provide an efficient method of filtering and sorting on the value of fields in work items. To make use of this functionality, you must first pre-designate the fields you want to filter/sort on as CDQP fields. Fields are designated as CDQP fields with the utility, **swutil**. This utility is used to create a list, on the TIBCO Process/iProcess Engine, of the case data fields that are available to use for filtering and sorting. For information about using **swutil**, see the *TIBCO iProcess swutil and swbatch Reference Guide*.

For information about using CDQP fields for filtering, see the *Using Case Data Queue Parameter Fields* section in the appropriate *Filtering Work Items and Cases* chapter on page 212, page 238, and page 264.

For information about using CDQP fields for sorting, see "Using Case Data Queue Parameter Fields" on page 277.

Passing Field Data when Keeping/Releasing Work Items

When keeping work items (**keepItems** method) or releasing work items (**releaseItems** method), you can specify which fields to send to the server. This allows you to limit the amount of field data that is being sent from the client to the server.

To specify which fields are to be sent to the server when keeping or releasing work items, do the following:

1. Construct an array of **vWIFieldGroup** objects, specifying the work item fields you would like sent to the server.
2. Pass the **vWIFieldGroup** object as an input parameter with the **keepItems** or **releaseItems** method.

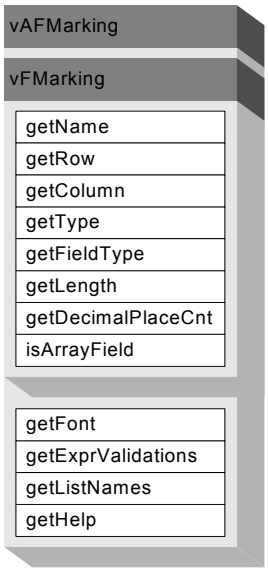
Note - If a field has been defined as a Numeric field on the form, and you are passing a value into that field when keeping or releasing the work item, you MUST pass in a Double, otherwise an exception is thrown.

What are Markings?

A “marking” is a field that is associated with a specific step on an iProcess Modeler-produced form. Placing the field on a form makes the association.

Note - Markings are applicable only for iProcess Modeler forms — they do not apply to other types of forms.

Markings are represented by the **vFMarking** and **vAFMarking** objects. These objects represent the marking as it is defined on a form at design time. They have properties that define the physical characteristics of the field/marking on the form, but they do not have a **getValue** method, i.e., markings do not contain data.



You can retrieve the **vAMarking** objects for an entire procedure or for a particular form:

- **Procedure** - Calling the **getFormMarkings** method on **sProcManager** returns an array of **vAFMarking** objects, one for each marking in the specified procedure.
- **Form** - Each form (**vForm** object) contains a **getFRows** method that returns an array of **vFRow** objects, one for each row on the form. The **vFRow** objects have a **getAFMarkings** method that returns an array of **vAFMarking** objects, one for each marking in that row.

The information available from these objects can be used to represent a form using other UI tools.

Type Validation on Fields/Markings

There are a couple of different *type* properties associated with fields and markings. These properties are accessible with the following methods:

- **vField.getType**
- **vFMarking.getFieldType**

Both of these methods return the type of data (e.g., numeric, date, ASCII text, etc.) that can be placed in the field/marking. These types are enumerated by the **SWFieldType** enumeration.

*Note - The **vFMarking** object also has a **getType** method. However, unlike **vField.getType**, it identifies the data-entry requirements for the marking on the form (optional, required, etc.). These are enumerated in the **SWMarkingType** enumeration.*

Note - Markings are applicable only for iProcess Modeler forms — they do not apply to other types of forms.

When the value of a field is set, the type of the variable passed is validated against the type specified by **vFMarking.getFieldType**. If the types do not match, the system will attempt to do a data conversion (details of what can be converted can be found in the **getValue** topic in the on-line help system). If the types do not match, and the data cannot be converted, a “type mismatch error” is generated.

Accessing Memo Fields

Fields of type **swMemo** can be accessed through the **getValue** method on the **vField** object (note that your TIBCO iProcess Objects Server must have CR 8427 implemented to be able to access memo fields). Also, all methods that have field names and field values as input parameters support the use of fields of type **swMemo** as input parameters.

Because the TIBCO iProcess Engine must perform file I/O to store memo data, the length of time between when the client requests that a memo value be stored on the server, and the time when the client receives a reply that the data was successfully stored, can be several seconds. Because of this, you may have a desire to configure the system so that if a specified period of time elapses waiting for a response from the server, the client will timeout and generate an error. To configure this "message wait time," you must add a Registry entry (Windows) or environment variable (UNIX), and set it to the number of milliseconds you would like the client to wait before timing out.

For information about setting the message wait time, see "Message Wait Time" on page 270.

Accessing Attachments

You cannot directly access the data in fields of type **swAttachment**. The data in this type of field is the TIBCO-generated name and path to the file on the server containing the data. You would have to locate the file, then read and parse the contents of the file yourself. They are not moved to the client machine.

Accessing System Fields

The built-in system fields (e.g., SW_CASE, SW_STARTER, etc.) provide the TIBCO iProcess Engine with references to information about work items and cases. These fields are primarily used when performing filtering and sorting functions.

The information that is available to the TIBCO iProcess Engine through the system fields is also available to the client through methods on various Value Objects. For example, **SW_CASENUM** is available to the client with the **vCaseId.getCaseNumber** method, **SW_PRIORITY** is available to the client with the **vWorkItem.getPriority** method, etc. The TIBCO iProcess Engine, however, doesn't have access to the Value Objects, so those methods can't be used in filter and sort expressions. Instead, you must use the system field names in your expressions. For example, when constructing the **vWICriteria** object, the *aFilterExpression* parameter could be set to the following string:

```
"SW_CASENUM=5"
```

This evaluates to a Boolean expression — the work items for which this is True (where case number equals 5) will be returned.

For lists of the system fields that can be used when filtering and sorting, see the appropriate *Filtering Work Items and Cases* chapter on page 208, page 234, or page 260 and "System Fields used in Sorting" on page 275.

Array Fields

Array fields are defined using TIBCO Business Studio's Field Definition dialog in the same way as standard single-instance iProcess fields. An option on the Field Definition dialog allows you to designate the field as either a single-instance field or an array field. If designated as an array field, the field can hold up to 99,999 data elements, each identified by an index (the field name followed by an index number in brackets "[ ]"). For example, the array field CUSTNAME would be referenced by:

```
CUSTNAME[0]
CUSTNAME[1]
CUSTNAME[2]
```

...and so on. See "Array Field Indexes" on page 173 for more information about indexes.

Array fields can be used in the same way as single-instance iProcess fields, i.e., they can be used on forms and in scripts. They are also used with both dynamic sub-procedure call steps and graft steps. These types of steps allow an arbitrary number of sub-procedure cases to be started from, or grafted to, a parent case. Array fields provide the ability to dynamically create variable length sets of data elements that may be passed between the parent and sub-procedures.

You can determine whether a field is a single-instance field or an array field using the following method:

- **isArrayField** - This Boolean flag, available on **vFieldDef** and **vFMarking** returns True if the field is defined as an array field. It returns False if the field is defined as a single-instance field.

When used with dynamic sub-procedure call steps and graft steps, array fields are used in the following ways:

- **To identify the sub-procedures to start** - When a dynamic sub-procedure call step or graft step is defined in a procedure, instead of specifying the names of the sub-procedures (or external processes) to start from that step, a text array field is specified. For dynamic sub-procedures, the customer application is responsible for assigning the names of the sub-procedures it wishes to have started to the elements of the array field. For graft steps, the startGraftTask method is called, which specifies the sub-procedures / external processes to start — these names are automatically written to the elements of the array field.

  For example, suppose a dynamic sub-procedure call step specifies SPROCS as the array field that will contain the names of the sub-procedures to start. If the application wants sub-procedures SUB1, SUB5, and SUB7 to be started, it must assign to the elements of SPROC the following prior to the step being processed:

  ```
  vField[] startProcs = new vField[3];
  startProcs[0] = new vField("SPROCS[0]", "SUB1", SWFieldType.swText);
  startProcs[1] = new vField("SPROCS[1]", "SUB5", SWFieldType.swText);
  startProcs[2] = new vField("SPROCS[2]", "SUB7", SWFieldType.swText);
  ```

  Note that the array field elements do NOT have to have contiguous indexes. For example, they could appear as follows::

  ```
  vField[] startProcs = new vField[3];
  startProcs[0] = new vField("SPROCS[0]", "SUB1", SWFieldType.swText);
  startProcs[1] = new vField("SPROCS[2]", "SUB5", SWFieldType.swText);
  startProcs[2] = new vField("SPROCS[3]", "SUB7", SWFieldType.swText);
  ```

They also don't have to be in order. For example:

```
vField[] startProcs = new vField[3];
startProcs[0] = new vField("SPROCS[0]", "SUB1", SWFieldType.swText);
startProcs[1] = new vField("SPROCS[2]", "SUB5", SWFieldType.swText);
startProcs[2] = new vField("SPROCS[1]", "SUB7", SWFieldType.swText);
```

- **To identify the start steps** - When a dynamic sub-procedure call step is defined in a procedure, a non-default "start step" may also be defined for each of the sub-procedures to be started (this functionality is not available for graft steps). These are also specified in a text array field. The step to start each sub-procedure is taken from the specified array field using the same element index as the "sub-procedure to start" array field.

  For example, continuing from the example in the bullet item above, suppose a dynamic sub-procedure call step specifies STARTSTP as the array field that will contain the names of the non-default start steps for the sub-procedures that are started by that dynamic sub-procedure call step. If the application wants sub-procedure SUB1 to start at STEP1, SUB5 to start at STEP2, and SUB7 to start at STEP5, it must assign to the elements of STARTSTP the following values prior to the step being processed:

```
vField[] startSteps = new vField[3];
startSteps[0] = new vField("STARTSTP[0]", "STEP1", SWFieldType.swText);
startSteps[1] = new vField("STARTSTP[1]", "STEP2", SWFieldType.swText);
startSteps[2] = new vField("STARTSTP[2]", "STEP5", SWFieldType.swText);
```

  If the "start step" array field (STARTSTP in this example) element that corresponds to the same index as the "sub-procedure to start" array field is unassigned, the sub-procedure case is started at that sub-procedure's default start step.

Array Field Indexes

As described in the above subsections, array field elements can be referenced using an index number enclosed in brackets. They may also be referenced using just the field name without a specified index. In this case, the value returned is dependent on two system fields that specify the array element that is to be used as the data source. These system fields are:

- **IDX_**<Array Field Name> - For example, if an array field is called CUSTNAME, its corresponding index system field is "IDX_CUSTNAME". Each array field has a corresponding index system field, which is automatically created when the array field is defined in TIBCO Business Studio. Whenever an array field is referenced by only its name without an index identifier, the index number from this system field is used (if it has been assigned).

  An example of using this index system field in a script to assign values to three elements of the array field CUSTNAME is shown below :

  IDX_CUSTNAME := 0
  CUSTNAME := "John Doe"
  IDX_CUSTNAME := 1
  CUSTNAME := "Jane Doe"
  IDX_CUSTNAME := 2
  CUSTNAME := "Danny Doe"

- **SW_GEN_IDX** - If the "IDX_<Array Field Name>" system field is not currently assigned for an array field, the array element index is taken from this generic index system field. This is use-

ful if the application requires several different array fields to hold data sets across fields with the same index, it can simply ensure that all of the individual system index fields are set to unassigned (SW_NA), then set SW_GEN_IDX to the desired index. See the example below:

IDX_CUSTNAME := SW_NA
IDX_ACCOUNT := SW_NA
SW_GEN_IDX := 0
CUSTNAME := "John Doe"
ACCOUNT := 11111
SW_GEN_IDX := 1
CUSTNAME := "Jane Doe"
ACCOUNT := 55555
SW_GEN_IDX := 2
CUSTNAME := "Danny Doe"
ACCOUNT := 77777

If neither the index system field for an array field, nor the generic index system field, are assigned, the index defaults to 0 (zero).

When an array field is "marked" on a form during procedure definition, it is identified only by its field name. No element index is specified.

Using Array Fields in Filter Expressions

Array fields can be used in filter expressions.

You can include array fields with an index in brackets in filter expressions when filtering cases (e.g., NAME[0] = “abcd”). Note, however, that the index value must be a constant (i.e., a single number); it cannot be a variable or expression.

Array fields with an index in square brackets cannot be used when filtering work items. When filtering work items, you can use array fields without an index — the WIS uses the default index number, either “IDX_<array_field_name>” or “SW_GEN_IDX”.

Requesting, Returning, and Setting All Array Field Elements

This section describes how to request, return, and set all of the array field elements at one time, rather than individually.

Requesting All Array Field Elements

As described in the previous subsections, individual array field elements can be requested by specifying the field name and the element index in the following format:

`FieldName[`*`ElementIndex`*`]`

However, the values of *all* of the array's elements can be requested by including an asterisk as the element index:

`FieldName[*]`

Note, however, that this method of requesting all array field elements is intended for use only when locking work items and when getting cases. The following table shows the methods in which `FieldName[*]` may be used, as well as the content objects (in which you specify the fields to return) used by those methods:

Method	Content Object
lockItems	vWIFGContent
getACases getACaseList makeACaseList	vACaseContent

You should *not* use `FieldName[*]` when calling the **getWorkItems** and **getWorkItemList** methods. When used with these methods, the expected array will not be returned.

Returned Array Field Elements

When you request that all array field elements be returned by using `FieldName[*]`, the requested array field is returned as a **vField** object. The vField object contains the following information:

`vField.Name` = *Name of array field*

`vField.Type` = *SWFieldType constant*

`vField.Value` = *Array of objects*

where:

- *Name of the array field* is the name of the field with no brackets.
- *SWFieldType constant* identifies the type of data in the array field elements. For a list of the possible **SWFieldType** constants when using array fields, see the "SWFieldType Enumerations Used With Array Fields" section below.
- *Array of objects* is a system array of the values in all of the elements of the array field. The array element with the largest index that has a value assigned is treated as the last element in the array. All data elements from zero to this element index are returned. For example, if the array field LASTNAME has been assigned the following values ...

  LASTNAME[0] = Miller
  LASTNAME[3] = Thomas
  LASTNAME[6] = Gunderson

  ... all elements 0 through 6 are returned. Elements 0, 3, and 6 will contain the values assigned to those elements — these values will be in the form of **java.lang.Object** or **java.util.Object** objects, whose type is determined by the field's assigned type. For example, if the field type is swArrayOfText, **vField.getValue** will return an array of system objects, with **java.lang.String** object values assigned to those elements that contain a value. Elements that have not been assigned a value will contain an **SWEmptyField** object. The type of java.lang.Object returned for each possible SWFieldType is shown in the table in the "SWFieldType Enumerations Used With Array Fields" section below.

For example, if all of the array field elements for a numeric array field named COUNT were requested, the returned **vField** would look as follows:

`vField.Name` = COUNT

`vField.Type` = swArrayOfNumeric

`vField.Value` = java.lang.Double[]

The array of system objects returned by **vField.getValue** will have **java.lang.Double** object values assigned to the array field elements (with possible **SWEmptyField** objects).

SWFieldType Enumerations Used With Array Fields

The table below shows the **SWFieldType** enumerations that are used with array fields:

Constant	Type of Array	Objects Returned
swArrayOfComma	Array of comma-separated numerics	java.lang.Double
swArrayOfDate	Array of dates	java.util.Date
swArrayOfMemo	Array of memos	java.lang.String
swArrayOfNumeric	Array of numerics (real numbers)	java.lang.Double
swArrayOfText	Array of ASCII text values	java.lang.String
swArrayOfTime	Array of times	java.util.Date

Setting All Array Field Elements

To set all elements of an array field:

1. Create an array of **java.lang.Object** or **java.util.Object** objects to represent the data that is to be set for the array field.
2. Assign the system array to the **Value** property of vField.
3. Set the **Type** property of vField to the corresponding **swArrayOfxxx** type listed in section "SWFieldType Enumerations Used With Array Fields".
4. Set the **Name** property of vField to the array field name (with no index specified).

Example

```
Object[] ArrayOfText = new Object[]{"Text1", "Text2", new SWEmptyField(), "Text3"};
Object[] ArrayOfNumeric = new Object[]{new Double(12.3), new SWEmptyField(), new
Double(2.5)};
Object[] ArrayOfDate = new Object[]{new Date(2003 - 1900, 3 - 1, 3), new
SWEmptyField(), new Date(2004 - 1900, 4 - 1, 4), new Date(2005 - 1900, 5 - 1, 5)};
Object[] ArrayOfTime = new Object[]{new Date(2002 - 1900, 2 - 1, 1, 10, 10, 10), new
SWEmptyField(), new Date(2002 - 1900, 2 - 1, 1, 23, 23, 23)};
Object[] ArrayOfMemo = new Object[]{"Memo1 is a small memo.", "Memo2 is small too?",
new SWEmptyField(), "Memo3 is a smallest memo.", "Memo4 is smaller memo."};

vField[] Fields = new vField[]{
new vField(ArrayTextName + "[0]", new SWEmptyField(), SWFieldType.swText),
new vField(ArrayNumericName + "[0]", new SWEmptyField(), SWFieldType.swNumeric),
new vField(ArrayDateName + "[0]", new SWEmptyField(), SWFieldType.swDate),
new vField(ArrayTimeName + "[0]", new SWEmptyField(), SWFieldType.swTime),
new vField(ArrayMemoName + "[0]", new SWEmptyField(), SWFieldType.swMemo),
```

```
new vField(ArrayTextName, ArrayOfText, SWFieldType.swArrayOfText),
new vField(ArrayNumericName, ArrayOfNumeric, SWFieldType.swArrayOfNumeric),
new vField(ArrayDateName, ArrayOfDate, SWFieldType.swArrayOfDate),
new vField(ArrayTimeName, ArrayOfTime, SWFieldType.swArrayOfTime),
new vField(ArrayMemoName, ArrayOfMemo, SWFieldType.swArrayOfMemo)
};
```

Setting Array Field Values to SWEmptyField in SWEmptyField in XML

This section describes how to set array field elements to **SWEmptyField** when you are constructing XML (for example, using custom JavaScript).

Note that it is admissible to not include the Value attribute in the XML for a *non-array field*:

```
<sso:vField>
   <sso:Name>GRPS</sso:Name>
   <sso:FieldType>swText</sso:FieldType>
</sso:vField>
```

This works for a non-array field because the **Value** field in the **vField** object takes on its default value, which is SWEmptyField.

However, when you are using array fields, you *must* include the Value attribute in the XML — the array field elements do not take on a default value.

For example, assuming an array field with two elements, to set the value of those elements to SWEmptyField, the XML must look like the following:

```
<sso:vField>
   <sso:Name>GRPS</sso:Name>
   <sso:FieldType>swArrayOfText</sso:FieldType>
   <sso:Value>[com.staffware.sso.data.SWEmptyField]|
              [com.staffware.sso.data.SWEmptyField]</sso:Value>
</sso:vField>
```

This produces a **vField** object with a value that is an array of size two, with each element of the array having the value SWEmptyField (note that the value of an array field must be an array).

For more information about SWEmptyField, see "Uninitialized Fields" on page 167.

Date Format

Fields that contain a date, by default, use the format dd/mm/yyyy. This format is specified using characters 27-29 (*dmy*) of line 5 of the *SWDIR***etc\staffpms** (Windows) or $SWDIR/**etc/staffpms** (UNIX) file, as follows:

```
%2d/%2d/%4d\/\%s%s %s, %s\dmy\wdmy\%2d:%2d\:\ AM\ PM\Week\NYYYYYN
```

In addition, the first 11 characters determine how many characters to allow for each part of the date. The default is to use two characters for the day and month, and four characters for the year. You can change the order of these, but not the number of characters, i.e., the day and month must always be two characters, and the year must always be four characters.

Example 1:

To change the date format to mm/dd/yyyy:

```
%2d/%2d/%4d\/\%s%s %s, %s\mdy\wdmy\%2d:%2d\:\ AM\ PM\Week\NYYYYYN
```

Example 2:

To change the date format to yyyy/mm/dd:

```
%4d/%2d/%2d\/\%s%s %s, %s\ymd\wdmy\%2d:%2d\:\ AM\ PM\Week\NYYYYYN
```

Character Encoding

ICU conversion libraries can be used to specify the desired character encoding.

To use the ICU conversion libraries, you must create the following environment variable (UNIX systems) or Registry entry (Windows systems) and set it to the name of the converter you wish to use.

— TISOUnicodeConverterName

For more information about adding this environment variable or Registry entry, see "Character Encoding Using ICU Conversion Libraries" on page 293.

12

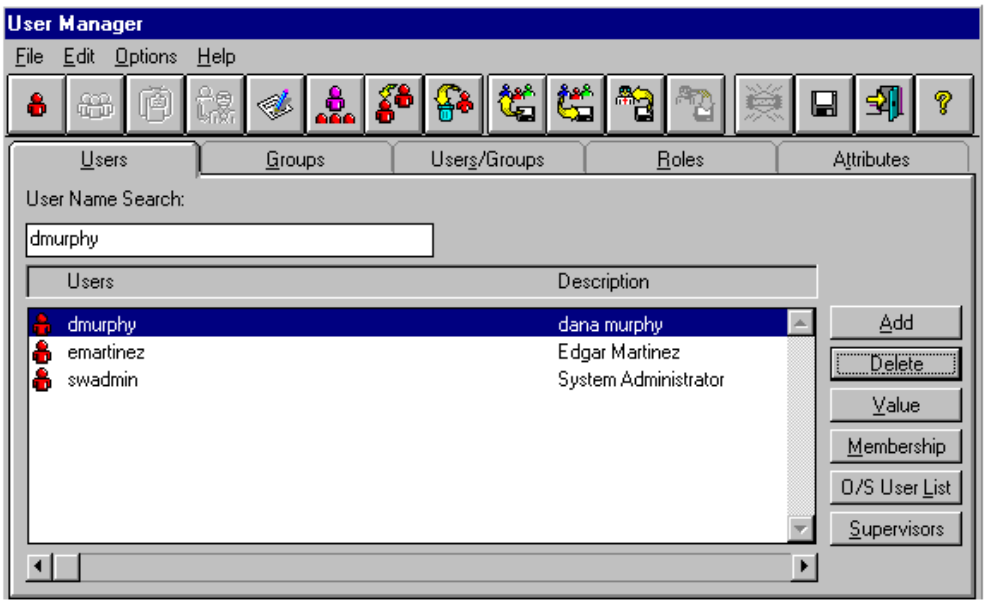
User Administration

Introduction

User administration tasks can be performed in two ways:

- using the User Manager, or
- through methods on TIBCO iProcess Server Objects.

The **User Manager** is part of the TIBCO Administration Managers suite of utilities. It is used to administer users, groups, roles, and attributes, which are all associated with iProcess users.



All of the user administration functions that can be performed using the User Manager can also be done using methods on TIBCO iProcess Server Objects. For information about using the User Manager, see the *TIBCO iProcess Workspace (Windows) Manager's Guide*.

The remainder of this chapter describes performing user administration tasks by using the methods on TIBCO iProcess Server Objects.

Types of Users

TIBCO iProcess Server Objects make use of the following types of users:

- **O/S User** - This is a user that has been created in the operating system. When creating an iProcess user (see below), the user may or may not have to be an existing O/S user, depending on

the value of a TIBCO iProcess Objects Server configuration parameter. See "Is an O/S User needed for every iProcess User?" on page 183 for more information.

- **iProcess User** - This is a user that has been created for the purpose of "using" a client application — this is the user whose user ID and password will be passed as parameters when a Server Object is constructed. An iProcess user is created using the **createUser** method on the **sNode** object.

MOVESYSINFO Function

Whenever you perform a function that affects a user, group, role, attribute, or queue supervisor definition, a **MOVESYSINFO** function must be performed to "commit" the change that you've made. There are a number of ways the MOVESYSINFO function can be performed:

- **Implicitly** - This means that the MOVESYSINFO function will be performed automatically, in background, after a user, group, role, attribute, or queue supervisor definition is changed. Note that this can tie up the background and WIS/WQS processes for long periods of time if there are lots of users.

  To specify that the MOVESYSINFO function is to be implicitly performed, set the **ImplicitMoveSysInfo** TIBCO iProcess Objects Server configuration parameter to 1 (UNIX), or check the appropriate box on the *TIBCO iProcess Objects Server Configuration Utility* **Users** tab (Windows).
- **Explicitly** - This means that the MOVESYSINFO function will NOT be performed automatically after a user, group, role, attribute, or queue supervisor definition is changed. Instead, the client application must make a method call to cause the MOVESYSINFO function to be performed. This allows you to more closely control this functionality.

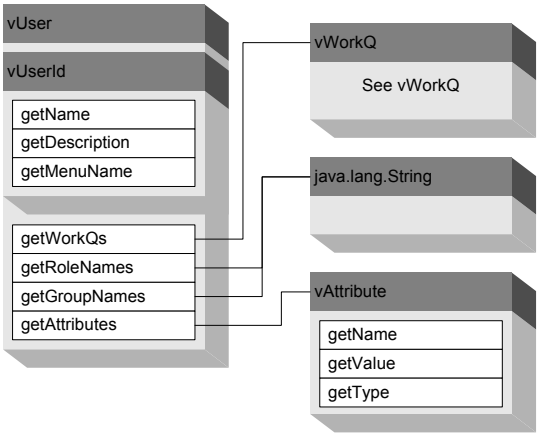
  To specify that the MOVESYSINFO function is to be explicitly performed, set the **ImplicitMoveSysInfo** TIBCO iProcess Objects Server configuration parameter to 0 (UNIX), or uncheck the appropriate box on the *TIBCO iProcess Objects Server Configuration Utility* **Users** tab (Windows). The actual MOVESYSINFO function can then be executed by calling the **moveSysInfo** method on **sNode**. (You must have system administrator authority (MENUNAME = ADMIN) to call the moveSysInfo method. See "User Attributes" on page 187 for information about the MENUNAME attribute.) See your on-line help for information about this method.
- **Using swutil** - If changes to a large number of users, groups, roles, attributes, or queue supervisor definitions need to be made, your best option might be to use the **swutil** "Update User Information" function:

  ```
  swutil USERINFO filename
  ```

  For more information, see the *TIBCO iProcess swutil and swbatch Reference Guide*.

For more information about the TIBCO iProcess Objects Server configuration parameters, see the *TIBCO iProcess Objects Server Administrator's Guide.*

iProcess Users



iProcess users are represented by the **vUserId** and **vUser** objects. These local Value Objects provide access to information about the user, such as the work queues assigned to the user, the groups the user is a member of, etc.

Each node (TIBCO iProcess Objects Server) maintains a list of iProcess users that have been created on that node. This list of users can be retrieved from the server by calling the following methods on **sNode**:

- **getUsers** - The method returns an array of **vUser** objects, one for each user specified by the *aUserNames* parameter.
- **getUserList** - This method returns a pageable list of **vUser** objects, one for each user defined on the node.
- **getUserListHeld** - This method returns a pageable list of users that had been previously held.

When **vUser** objects are retrieved from the server using one of the methods listed above, you can control whether or not dependent objects (**vWorkQ** objects, etc.) are also retrieved from the server by passing a **vUserContent** object with the method call (see "Retrieving Dependent Objects" on page 160 for information). Whether or not you will also retrieve dependent objects will depend on how you are going to use the **vUser** objects after they are retrieved.

Two special iProcess users are automatically created when the TIBCO iProcess Engine is installed:

- **swuser** - This user is used to run the Process Invocator service.
- The IPEADMIN user - The IPEADMIN user (for iProcess Engine Administrator) has complete access privileges. This user can be designated as any iProcess user when the TIBCO iProcess Engine is installed. It defaults to the user installing the iProcess Engine.

Creating an iProcess User

An iProcess user is created on a specific node with the **sNode.createUser** method.

*Note - The user being created may have to already exist in the operating system, depending on the setting of the **CheckOSUser** configuration parameter. See the "Is an O/S User needed for every iProcess User?" section below for more information.*

The **createUser** method allows you to optionally specify groups to add the user to, and attribute values to assign to the user when the user is created. Note that the user name should not exceed 23 characters. Doing so may result in errors from the TIBCO iProcess Objects Server when functions are performed against that user's work queue.

When an iProcess user is created, a corresponding directory for that user is added on the node at *SWDIR***queues***username* (Windows) or $SWDIR/**queues**/*username* (UNIX). If you are using a TIBCO Process Engine, this *username* directory will contain a **staffo** file, which contains all of the work item data sent to the user. If you are using a TIBCO iProcess Engine, the work item data is stored in the **staffo** database table.

Creating a user also causes both a “test” and “released” work queue for the user to be created. See “Test vs. Released Work Queues” on page 104 for more information.

You must have system administrator authority (MENUNAME = ADMIN) to create a user. See “User Attributes” on page 187 for information about the MENUNAME attribute.

Deleting an iProcess User

One or more iProcess users can be deleted from the node with the **sNode.deleteUsers** method.

When an iProcess user is deleted, that user’s corresponding directory at *SWDIR***queues***username* (Windows) or $SWDIR/**queues**/*username* (UNIX) is NOT deleted. If you have a desire to remove directories associated with deleted users, this must be done through the operating system.

Prior to deleting a user, an administrator should unlock any work items that the user may have locked. If the user is deleted, then another user attempts to unlock a work item that was locked by the deleted user, a “Work Queue not found” error message is returned.

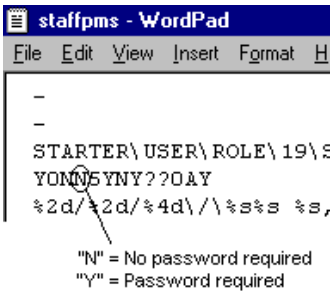
Important - Before deleting a user, ensure that there are no work items in the user’s work queue, or that that user is not the addressee of a step, because after the user is deleted, their work queue is no longer accessible

You must have system administrator authority (MENUNAME = ADMIN) to delete a user. See “User Attributes” on page 187 for information about the MENUNAME attribute.

Is an O/S User needed for every iProcess User?

The system can be configured to either require or not require that a user must first be a valid O/S user prior to that user being created on the system with the **createUser** method. When **createUser** is called, the iProcess Objects Server checks the following:

- **DISABLE_USER_CHECK** - If this engine process attribute is set to 1, the user does not have to be a valid O/S user before adding that user with the **createUser** method, regardless of the other settings described below.
- **CheckOSUser** - If this iProcess Objects Server configuration parameter is set to 0, the user does not have be a valid O/S user before adding that user with the **createUser** method. Note, however, that if you set this parameter to 0, you must also disable password checking in the engine (if password checking is not disabled, the **CheckOSUser** parameter is ignored). To enable/disable password checking, modify the $SWDIR/**etc/staffpms** (UNIX) or *SWDIR***etc\staffpms** (Windows) file. This is none by specifying a “Y” or “N” in the 4th character of the 4th line in the **staffpms** file.



Changing the User’s Password

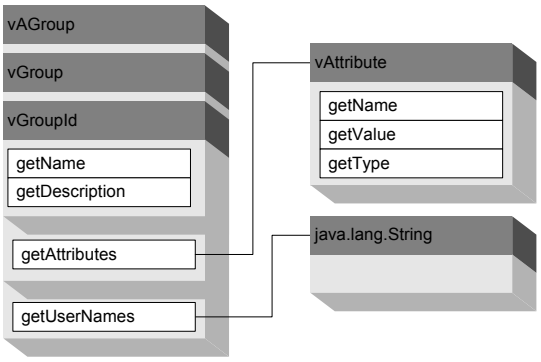
An iProcess user’s password can be changed with the **sUser.changePassword** method.

iProcess users can change *only* their own password. Even System Administrators cannot change another user’s password. The only way to change a password of another user is by using tools available through the operating system.

User Groups

A group represents a collection of iProcess users. For each group that is created, a work queue is automatically created with the same name as the group name. The purpose of the group work queue is to allow all users that are members of the group to work on the collection of work items in the group queue.

Groups are represented by the **vGroupId**, **vGroup**, and **vAGroup** objects. These local Value Objects provide access to information about a group, such as the attributes assigned to the group, and the names of the users belonging to the group.



A group is specific to the node (TIBCO iProcess Objects Server) on which it is created. Each node maintains a list of user groups that have been created on that node. This list of groups can be retrieved from the server by calling the following methods on **sNode**:

- **getAGroups** - The method returns an array of **vAGroup** objects, one for each group specified by the *aGroupNames* parameter.
- **getAGroupList** - This method returns a pageable list of **vAGroup** objects, one for each group defined on the node.
- **getAGroupListHeld** - This method returns a pageable list of groups that had been previously held.

When **vAGroup** objects are retrieved from the server using one of the methods listed above, you can control whether or not dependent objects (e.g., **vAttribute** objects, etc.) are also retrieved from the server by passing a **vAGroupContent** object with the method call (see "Retrieving Dependent Objects" on page 160 for information). Whether or not you will also retrieve dependent objects will depend on how you are going to use the **vAGroup** objects after they are retrieved.

The **sUser** object also has a **getGroups** method that retrieves from the server a list of all of the groups to which that specific user belongs.

Creating a User Group

Groups can be created on a specific node with the **sNode.createGroups** method.

The **createGroups** method has optional parameters that allow you to specify attributes/values for the group, and provide the names of users to be members of the new group. Note that the group name should not exceed 23 characters. Doing so may result in errors from the TIBCO iProcess Objects Server when functions are performed against that group's work queue.

When a user group is created, a corresponding directory for that group is added at *SWDIR***queues***groupname* (Windows) or $SWDIR/**queues**/*groupname* (UNIX). If you are using a TIBCO Process Engine, this *groupname* directory will contain a **staffo** file, which contains all of the work item data sent to the group. If you are using a TIBCO iProcess Engine, the work item data is stored in the **staffo** database table.

Creating a group also causes both a "test" and "released" work queue for the group to be created. See "Test vs. Released Work Queues" on page 104 for more information.

You must have system administrator authority (MENUNAME = ADMIN) to create a user group. See "User Attributes" on page 187 for information about the MENUNAME attribute.

Deleting a User Group

One or more groups can be deleted from the node with the **sNode.deleteGroups** method.

When a user group is deleted, its corresponding directory at *SWDIR***queues***groupname* (Windows) or $SWDIR/**queues**/*groupname* (UNIX) is NOT deleted. If you have a desire to remove directories associated with deleted groups, this must be done through the operating system.

Important - Before deleting a group, ensure that there are no work items in the group's work queue, or that that group is not the addressee of a step, because after the group is deleted, its work queue is no longer accessible.

You must have system administrator authority (MENUNAME = ADMIN) to delete a user group. See "User Attributes" on page 187 for information about the MENUNAME attribute.

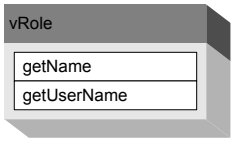
Adding and Removing Users to/from a Group

Once a user group is created with the **createGroups** method (or through the User Manager), you can add users to the group with the **sNode.addUsersToGroups** method. Users can be removed from a group with the **removeUsersFromGroups** method.

You must have system administrator authority (MENUNAME = ADMIN) to add or remove users to/from a user group. See "User Attributes" on page 187 for information about the MENUNAME attribute.

Roles

A role is a job title or function, such as Account Manager. One iProcess user is assigned to the role. Within a procedure, the addressee of a step can be specified as the role. That way if the person assigned to that job title changes, all you have to do is change the user assigned to the role. The procedure definition does not have to change.



Roles are represented by the **vRole** object. This local Value Object has methods that allow you to determine the name of the role and the name of the user assigned to that role.

Roles are specific to the node (TIBCO iProcess Objects Server) on which they are created. Each node maintains a list of the roles that have been created on that node. This list of roles can be retrieved from the server by calling the following method on **sNode**:

- **getRoles** - This method returns an array of **vRole** objects. You can either specify specific roles, or retrieve all roles that are defined on that node.

The **sUser** object also has a **getRoleNames** method that contains a list of all of the roles to which that specific user belongs.

Creating a Role

One or more roles can be created on a specific node with the **sNode.createRoles** method. This method requires that you construct an array of **vRole** objects, then pass that array as a parameter to the **createRoles** method.

You must have system administrator authority (MENUNAME = ADMIN) to create a role. See "User Attributes" on page 187 for information about the MENUNAME attribute.

Deleting a Role

One of more roles can be deleted from the node with the **sNode.deleteRoles** method. Note that there is no facility to delete all roles with one method call — you must delete each one individually.

You must have system administrator authority (MENUNAME = ADMIN) to delete a role. See "User Attributes" on page 187 for information about the MENUNAME attribute.

User Attributes

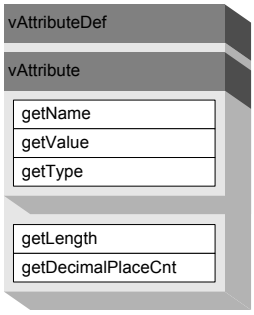
User attributes are properties/characteristics of an iProcess user or group. There are two types of user attributes:

- **Customizable** - You can create attributes that can hold any type of characteristic of the user/group. Some examples are an employee number, purchasing authority, etc.
- **Pre-defined** - Every iProcess user and group that is created is assigned each of the six pre-defined attributes shown in the table below.

Attribute	Description
DESCRIPTION	If the user or group was created with the **createUser** or **createGroups** method, this defaults to the name of the user or group. If the user or group was created with the User Manager, this is the value that was entered in the Description field.
LANGUAGE	This specifies the language in which user messages are displayed. It defaults to the language that is set in the regional settings on the system on which the user or group is created.
MENUNAME	This specifies the user's access authority (the name is derived from "the *menus* to which the user has access"). This attribute is only applicable to users (not groups). The possible values are: • **USER** - This is for ordinary users. With this access authority, the user can access work queues and start cases. This is the default value. • **MANAGER** - This has no affect on access authority — it is the same as USER. (If using the Work Queue Manager, this gives access to Case Administration for the purpose of viewing an audit trail.) • **PRODEF** - This is for procedure definers. This user has the authority to access TIBCO Business Studio for the purpose of defining procedures. • **ADMIN** - This is the system administrator authority. Users with this MENUNAME have authority to perform administrative-type functions. See "User Authority" on page 190 for a list of the functions a system administrator can perform. Note that the user's MENUNAME attribute is also available with the **vUserId.getMenuName** and **vActiveUser.getMenu** methods. *Note - You cannot change the value of this attribute for the IPEADMIN user (see page 182).*
QSUPERVISORS	This attribute specifies the users who can supervisor this user's/group's work queue. This is for the purpose of performing participation and redirection functions for the work queue. The list of queue supervisors is also available with the **getSupervisorNames** method. See "Work Queue Supervisors" on page 134 for more information.

Attribute	Description
SORTMAIL	Specifies the sequence in which work items are sorted in the user's or group's work queue. The possible values are: • **PROCEDURE** - Work items are sorted by Case Reference number. This is the default. • **ASCENDING ARRIVAL** - Work items are listed according to their arrival time — oldest first, followed by newest. • **DESCENDING ARRIVAL** - Work items are listed according to their arrival time — newest first, followed by oldest. • **ASCENDING DEADLINE** - Work items with deadlines are listed first (in order by: expired, first to expire, last to expire), followed by work items without deadlines. • **DESCENDING DEADLINE** - Work items without deadlines are listed first, followed by work items with deadlines (in order by: last to expire, first to expire, expired).
USERFLAGS	This attribute is used in conjunction with the step's forward permission (defined in the step definition) to determine if work items can be forwarded from this user's work queue: • " " - (Empty string) Work items from this user's work queue can be forwarded if the step's forward permission has been set. This is the default value. (This is called **Step Forward** in the User Manager.) • "**F**" - Any work item from this user's work queue can be forwarded, even if the step's forward permission has not been set. (This is called **Forward Any** in the User Manager.) • "**R**" - Work items from this user's work queue cannot be forwarded, even if the step's forward permission is set. (This is called **Forward None** in the User Manager.)

Attributes are represented by the **vAttribute** and **vAttributeDef** objects.



Attributes are specific to the node (TIBCO iProcess Objects Server) on which they are defined. All users and groups on that node take on those attributes (but each user might have different values for the attributes). When a new user or group is created on the node, they automatically acquire the attributes that are defined on the node.

Each node maintains a list of the attributes that have been defined on that node. This list of attributes can be retrieved from the server by calling the following methods:

- **sNode.getUserAttributes** - This method retrieves an array of **vAttribute** objects, one for each attribute defined for the specified user.
- **sNode.getGroupAttributes** - This method retrieves an array of **vAttribute** objects, one for each attribute defined for the specified group.
- **sNode.getAttributeDefs** - This method retrieves an array of **vAttributeDef** objects. You can either request specific attributes, or request all attributes defined on the node.
- **sUser.getAttributes** - This method retrieves an array of **vAttribute** objects, one for each attribute defined for the user represented by the **sUser** object.

You can also get the attributes that have been defined for a particular user or group from their respective local Value Objects, as follows:

- **vUser.getAttributes**
- **vGroup.getAttributes**

To be able to use these methods, of course, the dependent **vAttribute** objects must have been included when the local Value Objects were retrieved from the server (using the **vUserContent** and **vAGroupContent** objects).

Because the MENUNAME attribute is needed often to determine the user's access authority, the value of this attribute for the user is directly available with the following methods:

- **vUserId.getMenuName**
- **vActiveUser.getMenu**

Modifying an Existing Attribute Value

The following method can be used to modify or assign a value to an existing attribute:

- **sNode.changeAttributes** - This method allows you to change one or more attribute values for one or more users.

This method requires that you construct an array of **vAttribute** objects, one for each attribute you want to modify, and pass it as a parameter to the method call.

You must have system administrator authority (MENUNAME = ADMIN) to modify an attribute value. See "User Attributes" on page 187 for information about the MENUNAME attribute.

Creating an Attribute Definition

An attribute definition is created on a specific node with the **sNode.createAttributeDefs** method.

When an attribute is added to the node, it is automatically assigned to each existing user and group on that node.

You must have system administrator authority (MENUNAME = ADMIN) to create an attribute definition. See "User Attributes" on page 187 for information about the MENUNAME attribute.

Deleting an Attribute

One or more attributes can be deleted from the node with the **sNode.deleteAttributeDefs** method.

This method is an exception to the way error handling is performed; if an error is detected in one or more of the attribute names passed in, none of the attributes specified are deleted, and there is no information about which parameters were in error in **vException.getExceptionDetails**. The message "Attribute not found" is returned.

You must have system administrator authority (MENUNAME = ADMIN) to delete an attribute definition. See "User Attributes" on page 187 for information about the MENUNAME attribute.

Caution - The system will allow you to delete the pre-defined attributes that it needs to function properly.

Why isn't the new User, Group, Role or Attribute Available?

iProcess users, groups, roles, and attributes are created by the background process, so there is a delay between when the method is called and when the new user, group, role, or attribute is available.

User Authority

Throughout this document references are made to the user/access authority you need to perform particular functions. This section summarizes these authorities.

There are two primary administrator-level authority designations:

- **System Administrator Authority** - This authority allows the user to perform administrative-type functions that the typical user would normally not be able to perform, such as creating/removing users, closing/purging cases, etc. See below for comprehensive lists of the functions you can perform with system administrator authority.

  A user is given system administrator authority by setting their MENUNAME attribute to ADMIN. This is done using the **changeAttributes** method on **sNode**. See "User Attributes" on page 187 for more information about user attributes.

  Note - To ensure that there is always a user that has system administrator authority, there is a special IPEADMIN user. The IPEADMIN user (for iProcess Engine Administrator) can be designated as any iProcess user when the TIBCO iProcess Engine is installed. It defaults to the user installing the iProcess Engine.
- **Case Administration Authority** - This authority allows the user to perform functions that are specific to cases of a procedure, such as viewing lists of cases, rebuilding a case, auditing cases, etc. See below for comprehensive lists of the functions you can perform with Case Administration authority.

  A user is given Case Administration authority as part of the procedure definition (using TIBCO Business Studio). Note that by default, when a procedure is defined in TIBCO Business Studio, *everyone* is given Case Administration authority unless you specifically give certain users Case Administration authority for that procedure; then only those users have that authority. See "Determining who can Audit Cases of a Procedure" on page 69 for information about how users are given this authority for a procedure.

The following tables list functions that can be performed with each user authority.

- You must have **system administrator** authority to perform the following functions:

Function	Methods
Close cases	closeCases closeCasesByCriteria
Purge cases	purgeCases purgeCasesByCriteria purgeAndReset
Create/delete users	createUser deleteUsers

Function	Methods
Create/delete groups	createGroups deleteGroups
Modify group membership	addUsersToGroups removeUsersFromGroups
Create/delete/modify attributes	createAttributeDefs deleteAttributeDefs changeAttributes
Create/delete roles	createRoles deleteRoles
Unlock work item locked by another user *(Note - Any user can unlock a work item that they have locked.)*	unlockItems
Add/remove queue supervisors	addSupervisors removeSupervisors
Change TIBCO iProcess Objects Server log settings	resetSrvLog setSrvLogOptions
Forward work items from other user's work queues *(Note - For information about the permission requirements to forward work items from your own work queue, see "Manually Forwarding Work Items" on page 131.)*	forwardItems
Move system information (iProcess Engine MOVESYSINFO function)	moveSysInfo

- You must have **case administration** authority to perform the following functions:

Function	Methods
Retrieve audit data for cases of the procedure	getAuditSteps

- You must have either **system administrator** or **case administration** authority to perform the following functions:

Function	Methods
Retrieve cases for any procedure on the node	getACases getACaseList
Retrieve case data for any procedure	getCaseFields
Get the filtered case count for any procedure	getCaseCnt

User Preference Data

The TIBCO iProcess Server Objects (Java) object model contains objects and methods that allow you to set, get, and delete user preference data as a text string in the iProcess Engine database.

The following object and methods provide this functionality:

- The **vPreference** object holds a name/value pair that represents the user preference data. This object must be constructed, then passed in the **setUserPreference** method to save a user's user preference data. This object is returned by the **getUserPreference** method. This object contains the following methods:
  - **getName** - Returns the name (key) given to the preference data when it is saved in the database.
  - **getValue** - Returns the user preference data.
- The **sNode**, **xNode**, **sUser**, and **xUser** objects contain the following methods to support this functionality:
  - **setUserPreference** - Saves the user preference data in the database. You must pass in the name of the user for whom you are saving user preference data, and a **vPreference** object, which contains a preference name (key) and the data.
  - **getUserPreference** - Returns a **vPreference** object containing the user preference data. You must pass in the name of the user and the preference name (key) to identify the user preference data in the database.
  - **deleteUserPreference** - Deletes the specified user preference data from the database.

The length of the user preference data (*aValue* on the **vPreference** object) is limited to 128K because of database limitations. There is no limitation on the number of uniquely named preferences (*aName* on the **vPreference** object) for a given user.

Note that you should not use a preference key (i.e., *aName* on the **vPreference** object) that begins with "com.tibco.bpm", as user preference keys beginning with those characters are reserved for use by TIBCO products.

Users can modify their own user preference data (using the methods on **sUser**/**xUser**). Only administrators can modify user preference data for other users (using the methods on **sNode**/**xNode**).

13

Filtering Work Items and Cases

Without Filtering Enhancements

Important - Read this page first to determine which of the *Filtering Work Items and Cases* chapters you should use.

Over time, enhancements have been made to the TIBCO iProcess Objects Server to improve the efficiency of filtering and sorting work items and cases. Because the scope of the enhancements is fairly major, three chapters are now provided in this guide that describe how filtering and sorting work, depending on which of the enhancements have been implemented in your TIBCO iProcess Objects Server. Use the table below to determine which chapter to use, based on the enhancements in your TIBCO iProcess Objects Server.

Note - Although the topic of sorting is covered in a separate chapter, filtering and sorting is described as a single process in the Filtering Work Items and Cases chapters because that is the way it is performed — work items or cases are filtered, then the result set from the filter operation is sorted.

Two major enhancements have been added to the TIBCO iProcess Objects Server that impact filtering and sorting:

- **WIS Work Item Filtering** - This enhancement moved all work item filter processing to the Work Item Server (WIS). With this enhancement, all of the additional capabilities previously provided by the TIBCO iProcess Objects Server can now be performed by the WIS when filtering work items (such as allowing the OR logical operator, allowing the <, >, <=, >=, and <> operators, etc.). Since the WIS has the work items cached, and has direct access to case data, this provides for very efficient filtering and sorting of work items.

  Your server/engine must have the following CRs implemented for this enhancement: TIBCO iProcess Objects Server - CR 12744; TIBCO Process/iProcess Engine - CR 12686.

- **Database Case Filtering** - This enhancement moved all case filter and sort processing to the database. With this enhancement, the filter expression is translated into an SQL select statement, which is used to create the result set from the cases in the database. The result set is then sorted. Because of the indexing ability of the database, this provides for very efficient filtering and sorting of cases.

  This enhancement was implemented in the following CRs: TIBCO iProcess Objects Server - CR 13182; TIBCO Process/iProcess Engine - CR 13098.

Use the following table to determine which of the *Filtering Work Items and Cases* chapters to use:

If your TIBCO iProcess Objects Server includes...	Use this chapter...
Neither of the enhancements listed above	Chapter 13
Only the WIS Work Item Filtering enhancement (CR 12744)	Chapter 14
Both the WIS Work Item Filtering and the Database Case Filtering enhancements (CRs 12744 and 13182)	Chapter 15

Introduction

The TIBCO iProcess Server Objects provide the ability to *filter* work items and cases, allowing you to filter out all those you aren't currently interested in. For example, you may only be interested in the work items that arrived in the work queue today, in which case you could specify a filter expression that filters out all work items other than those that arrived today:

```
"SW_ARRIVALDATE = !08/02/2001!"
```

The benefits of this are two-fold:

- It allows you to display to the user only those cases or work items that are of interest to them.
- It reduces the amount of work the client and the server need to do. When the result set from the filter operation results in fewer work items or cases, this reduces the work load on the client and server.

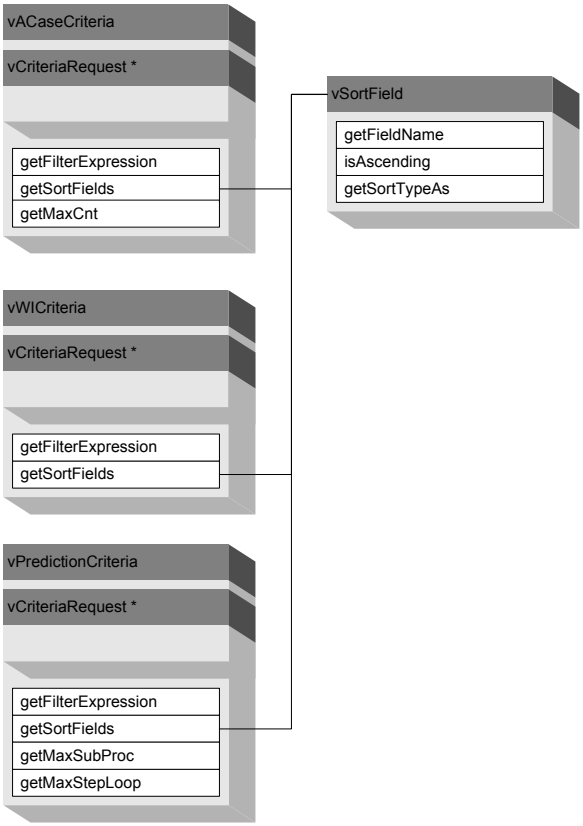
Specifying Filter Criteria

The TIBCO iProcess Server Objects provide three "criteria" objects that are used to specify filter criteria:

- **vACaseCriteria** (for cases)
- **vWICriteria** (for work items)
- **vPredictionCriteria** (for predicted work items) (Note that since predicted items are stored in the database, they are filtered in the same way as cases when you have the database case filtering enhancement — see *Filtering Work Items and Cases* on page 247 for more information.)

Note that these criteria objects are used for both filtering and sorting work items and cases — see "Sorting Work Items and Cases" on page 273 for information about sorting.

Work items and cases that can be filtered are always returned in pageable lists (**sPage-ableList**, **sPageableListR** or **sPageableListJ** objects) — see "Working with Lists" on page 137 for information about using pageable lists after the filtered work items or cases have been returned from the server.



The methods that return filtered work items and cases are summarized in the table below.

Method	Uses this Criteria Object	Returns Pageable List of this Object
sWorkQ.getWorkItemList	vWICriteria	vWorkItem
sWorkQ.getAWorkItemList	vWICriteria	vAWorkItem
sCaseManager.getACaseList	vACaseCriteria	vACase
sCaseManager.getPredictedItemList	vPredictionCriteria	vPredictedItem

To specify filter criteria for a pageable list of work items or cases, follow these steps:

1. Construct a **vACaseCriteria**, **vWICriteria**, or **vPredictionCriteria** object, setting the *aFilterExpression* parameter to the desired filter expression string — see the "Defining Filter Expressions" section below for specifics about creating these strings.
2. Pass the **vACaseCriteria**, **vWICriteria**, or **vPredictionCriteria** object as an input parameter with one of the methods listed in the table above, depending on the type of object you want returned in the pageable list.

Defining Filter Expressions

To filter work items, cases, or predicted work items, you must define a *filter expression* string, which is passed in the *aFilterExpression* parameter with the applicable criteria object (see the previous subsection). The filter expression string is evaluated against each work item or case you are requesting, returning either True or False. If it returns True, the work item/case is included in the pageable list; if it returns False, the work item/case is not included in the pageable list. For example, you may choose to filter the list to include only work items with the urgent flag set ("SW_URGENT = True").

Filter expression strings can contain elements such as system fields (SW_CASENUM, SW_NEW, etc.), logical operators (AND, OR), comparison operators (=, <, <=, etc.), ranges of values, etc. Details about filter expressions is described in the subsections that follow.

Note that the left and right side of comparison operators (=, <, >, <=, >=, <>, ?) must each consist of only a single field name or single constant. It cannot be an expression containing operators (+, -, /, *, etc.).

Example 1:
To define a filter for work items matching all new items from procedure LOANS, set the filter expression to the following string:

```
"SW_NEW=1 AND SW_PRONAME=\"LOANS\""
```

Example 2:
To define a filter for all work items that arrived after June 20, 2001 (assuming mm/dd/yyyy date locale setting in the engine), set the filter expression to the following string:

```
"SW_ARRIVALDATE > !06/20/2001!"
```

Example 3:

To define a filter for all cases with field LOAN_AMT having a value greater than 100000, set the filter expression to the following string:

```
"LOAN_AMT > 100000"
```

Number of Cases or Work Items in a Filtered Pageable List

There are a number of methods available that provide information about the number of cases or work items on a filtered pageable list. They are:

On **sPageableList**:

- **Available Count (getAvailableCnt) -** This returns the total number of items available in the indexed collection on the TIBCO iProcess Objects Server. This count includes only the work items or cases that satisfy the filter criteria.
- **Local Count (getLocalCnt) -** This returns the number of cases or work items currently being held in the local bock(s) on the sPageableList. If **isKeepLocalItems** has been set to False, this count will always be less than or equal to the number of items per block (**getItemsPerBlock**).

Note - The counts above require an understanding of how an indexed collection is created on the TIBCO iProcess Objects Server, and how blocks of objects are held locally in the pageable list. See *"Working with Lists" on page 137* *for information.*

On **vSummary**:

- **Exclude Count (getExcludeCnt) -** This returns the number of cases or work items that did not satisfy the specified filter criteria, and therefore, were not included in the pageable list.
- **Invalid Count (getInvalidCnt) -** This returns the number of cases or work items that did not satisfy the filter criteria because the filter expression was invalid within the context of the item. For example, the filter expression references a field name that is not defined on all work items or cases.
- **Over Maximum Count (getOverMaxCnt) -** This returns the number of cases that were *not* returned from the server because the number returned was limited using the *aMaxCnt* parameter on the **vACaseCriteria** constructor when retrieving a list of cases. This is applicable only when retreiving cases.

On **vWISummary**:

These counts are applicable only if the pageable list contains work items.

- **Urgent Count (getUrgentCnt) -** This returns the number of work items on the pageable list that are marked as urgent. A work item is marked as urgent if its priority value (vWorkItem.getPriorty) is less than or equal to a specific value. By default, the value is 10, although it can be modified in the **staffcfg** file.
- **Deadline Count (getDeadlineCnt) -** This returns the number of work items on the pageable list that have deadlines.
- **Unopened Count (getUnopenedCnt) -** This returns the number of work items on the pageable list that have not been opened (locked).

Filtering/Sorting in an Efficient Manner

The way in which you write your filter expressions can have an affect on how efficiently they are evaluated. This section provides guidelines about what types of elements you can include in your filter expressions (and those you should avoid) to ensure an efficient filter operation.

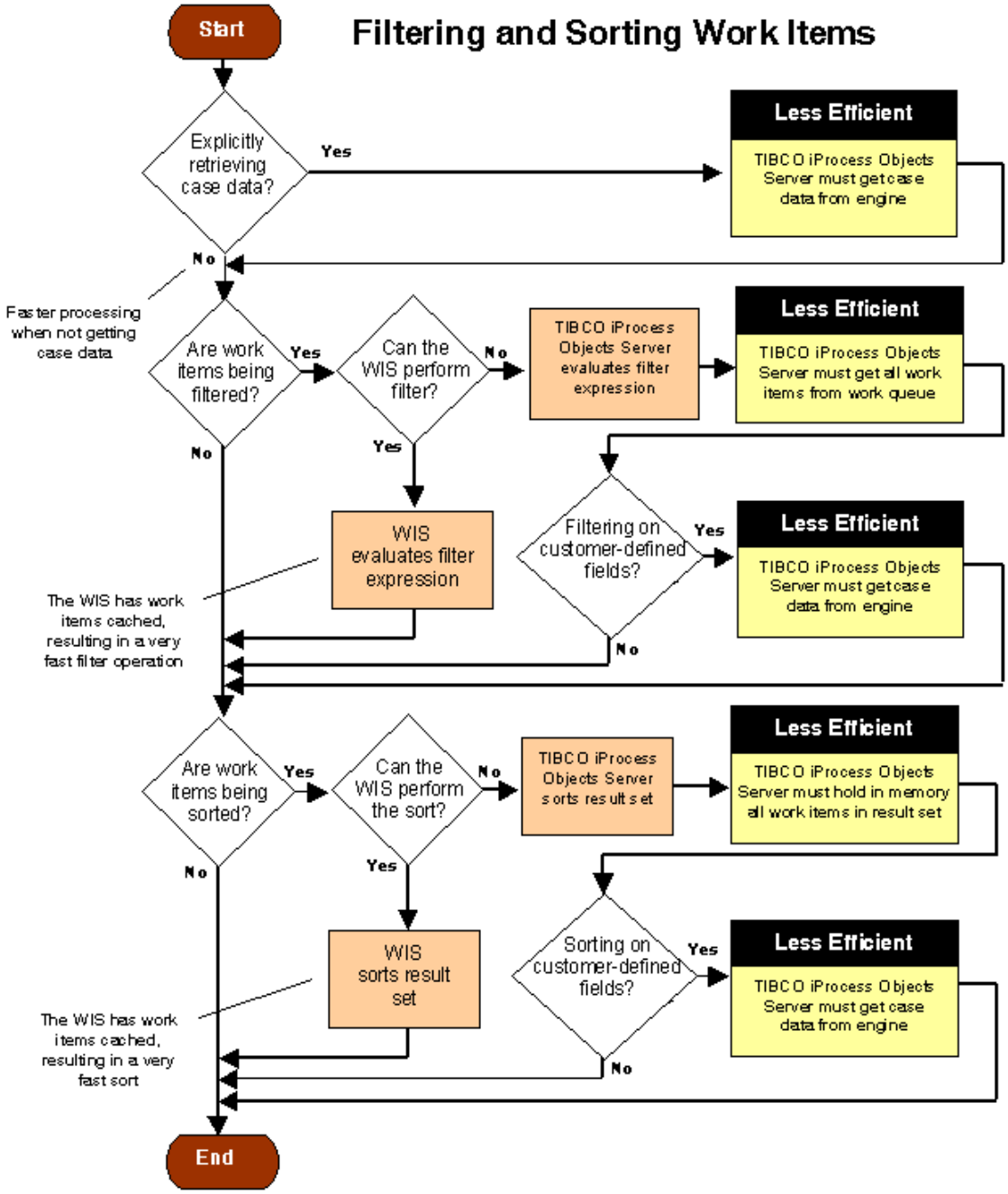
Flow diagrams (one for work items; one for cases) are shown in the following subsections that illustrate the decision process that takes place during a filter/sort operation. Note that the flow diagrams show filtering and sorting taking place in a single operation; that is the way filtering and sorting is processed — works items or cases are filtered to create a result set, then the result set is sorted. The flow diagrams also illustrate how to prevent the filter/sort operation from being less efficient.

Filtering/Sorting Work Items

When filtering and sorting work items:

- Work items can be filtered by either the WIS or the TIBCO iProcess Objects Server, depending on how you write your filter expressions. Ideally, you should write them so they are evaluated by the WIS because the WIS has work items cached in memory, allowing it to evaluate filter expressions for work items very quickly. The TIBCO iProcess Objects Server, however, provides additional filter criteria that the WIS does not provide. Using this additional criteria causes the expression to be evaluated by the TIBCO iProcess Objects Server, which requires that the server retrieve all of the work items from the queue to perform the evaluation, making the filter operation less efficient. See the tables in "Can the WIS Perform the Filter Operation?" on page 199 for a list of the filter expression elements that can be evaluated by the WIS and the TIBCO iProcess Objects Server, respectively.
- If you "get case data" in your application, this causes the filter processing to be less efficient because the TIBCO iProcess Objects Server must retrieve case data from the TIBCO iProcess Engine. See "Getting Case Data" on page 204 for more information.
- Work items can be sorted by either the WIS or the TIBCO iProcess Objects Server, depending on how you specify the sort criteria. It's preferable to have the WIS sort the result set from the filter operation, because if the TIBCO iProcess Objects Server performs the sort, it must hold all of the work items in the result set in memory. See "Can the WIS Perform the Sort Operation?" on page 201 for more information.

The diagram shown below illustrates the decision process that takes place when filtering and sorting work items.



As shown in the illustration, there are three actions that will cause the filter/sort operation to be less efficient when filtering and sorting work items:

- Getting case data
- Performing the filter operation on the TIBCO iProcess Objects Server
- Performing the sort operation on the TIBCO iProcess Objects Server

Additional information about these actions is provided in the subsections that follow.

Getting Case Data

Causing the server to "get case data" means that the TIBCO iProcess Objects Server must retrieve case data from the TIBCO iProcess Engine. This significantly slows down the filter/sort processing. The following actions cause the TIBCO iProcess Objects Server to get case data:

- Explicitly Retrieving Case Data - If you explicitly ask for case data using the *aCaseFieldNames* parameter when constructing a work item "content" object (**vWIContent**), the server will explicitly retrieve the data in those fields from the engine. There is always a performance hit when you ask for case data in this way, whether the filter/sort operation is performed by the WIS or the TIBCO iProcess Objects Server. See "Retrieving Field Data from the Server" on page 167 for information about the use of the *aCaseFieldNames* parameter.
- Having the TIBCO iProcess Objects Server filter on customer-defined fields - The TIBCO iProcess Objects Server does not have direct access to case data. Therefore, if your filter expression contains a customer-defined field (i.e., any field on a form that is not a system field (SW_PRIORITY, SW_PRONAME, etc.)), it must retrieve the data in that field from the TIBCO iProcess Engine, adversely affecting performance.
- Having the TIBCO iProcess Objects Server sort on customer-defined fields - The TIBCO iProcess Objects Server does not have direct access to case data. Therefore, if you sort on a customer-defined field (i.e., any field on a form that is not a system field (SW_PRIORITY, SW_PRONAME, etc.)), it must retrieve the data in that field from the TIBCO iProcess Engine, adversely affecting performance.

Note that although the flow diagram shows that there are three different places where you can take a performance hit by getting case data, the actual hit only occurs once, i.e., you don't take two performance hits, for instance, if you filter on customer-defined fields and sort on customer-defined fields; the TIBCO iProcess Objects Server only has to get case data once for the entire operation.

Can the WIS Perform the Filter Operation?

Work items can be filtered by either the WIS or the TIBCO iProcess Objects Server, depending on how you write your filter expression). If your filter expression contains only WIS-compatible criteria, the WIS will evaluate the expression. If your filter expression contains any of the expanded filter criteria provided by the TIBCO iProcess Objects Server, the TIBCO iProcess Objects Server will evaluate the expression.

The WIS has work items cached in memory, allowing it to evaluate filter expressions for work items very quickly. If the TIBCO iProcess Objects Server must perform the filtering operation, **it must retrieve all work items from the work queue**. Depending on the number of work items in the work queue, this can have a very significant impact on the performance of the filtering operation.

The following tables lists the elements that can be included in your work item filter expressions. If your filter expression contains any of the additional criteria provided by the TIBCO iProcess Objects Server, the entire expression is evaluated by the TIBCO iProcess Objects Server.

Filter Criteria the WIS can Evaluate

Element	Description
Comparison Operator	=
Logical Operator	AND
System Fields	System fields that are "WIS-compatible" (see the *WIS-compatible* columns in the table of system fields used for filtering — page 208) (They must also be applicable to filtering work items — see the *Applies To* column.)
Case Data Fields	Case data fields that have been defined as CDQPs. See "Filtering on Case Data Fields" on page 212 for information.
Wildcards[a]	The wildcard characters '*' and '?' as part of a string on equality checks. The '*' character matches zero or more of any character. The '?' character matches any single character.
Ranges of Values	Ranges of values can be included in your work item filter expressions by using a specific syntax — see "How to Specify Ranges of Values" on page 217 for information.

a. If the entire expression is WIS-compatible, the '*' and '?' are both interpreted as wildcards, as described above. However, if any part of the expression is NOT WIS-compatible, the '*' and '?' characters are interpreted literally by the TIBCO iProcess Objects Server (i.e., as asterisks and question marks), with the following exception — if the expression contains a '?' equality operator (e.g., SW_CASENUM ? "1*"), that part of the expression is evaluated separately. The part of the expression that contains the '?' equality operator CAN include the '*' and '?' wildcard characters (as in the SW_CASENUMBER ? "1*" example).

The TIBCO iProcess Objects Server can evaluate the filter criteria listed in the table above, as well as those listed in the table below.

Additional Filter Criteria the TIBCO iProcess Objects Server can Evaluate

Element	Description
Comparison Operators	<, >, <=, >=, <> (The ? character can also be used as an equality operator with regular expressions — see "Using Regular Expressions" on page 215.)
Logical Operator	OR
System Fields	System fields that are NOT "WIS-compatible" (see the *WIS-compatible* columns in the table of system fields used for filtering — page 208) (They must also be applicable to filtering work items — see the *Applies To* column.)
Case Data Fields	Case data fields that have NOT been defined as CDQPs. See "Filtering on Case Data Fields" on page 212 for information.
Regular Expressions	Regular expressions can be used when filtering work items, allowing you to do complex pattern matching. See "Using Regular Expressions" on page 215.

The following example filter expression can be evaluated by the WIS because it contains the '=' equality operator, and the SW_PRIORITY system field is WIS-compatible:

```
"SW_PRIORITY = 50"
```

The following example filter expression must be evaluated by the TIBCO iProcess Objects Server because it contains the '>' comparison operator:

```
"LOAN_AMT > 100000"
```

Work Item Server vs. TIBCO iProcess Objects Server Example

The following example illustrates the efficiency differences between the Work Item Server and the TIBCO iProcess Objects Server evaluating a filter expression. These filter expressions were run with 3,000 work items:

- "SW_CASENUM = 1 OR SW_CASENUM = 90" - This is evaluated by the TIBCO iProcess Objects Server because of the OR in the expression. It ran in 1450 ms.
- "SW_CASENUM = [1 | 90]" - This is evaluated by the Work Item Server. It ran in 20 ms.

Can the WIS Perform the Sort Operation?

Work items can be sorted by either the WIS or the TIBCO iProcess Objects Server, depending on the sort criteria you use. Whenever possible, you should use the sort criteria that can be evaluated by the WIS. If the TIBCO iProcess Objects Server must perform the sort operation, **it must hold in memory all work items in the filter result set**. If the result set from the filter operation is very large, this can consume a significant amount of memory.

The table below shows the sort criteria you can use to cause the sort operation to be performed by the WIS. It also lists the expanded criteria available by the TIBCO iProcess Objects Server. Using this expanded criteria causes the sort operation to be performed by the TIBCO iProcess Objects Server, which is less efficient because it must hold the result set in memory.

Sort Criteria the WIS can Process
• System fields that are "WIS-compatible". See the WIS-compatible column in the table of *System Fields used in Sorting* on page 275. (The system fields must be applicable to filtering work items.) • Case Data Queue Parameter (CDQP) fields. See "Sorting on Case Data Fields" on page 277 for more information.
Sort Criteria the TIBCO iProcess Objects Server must Process
• System fields that are NOT "WIS-compatible". See the WIS-compatible column in the table of *System Fields used in Sorting* on page 275. (The system fields must be applicable to filtering work items.) • Case data fields that have NOT been designated as Case Data Queue Parameter (CDQP) fields. See "Sorting on Case Data Fields" on page 277 for more information.

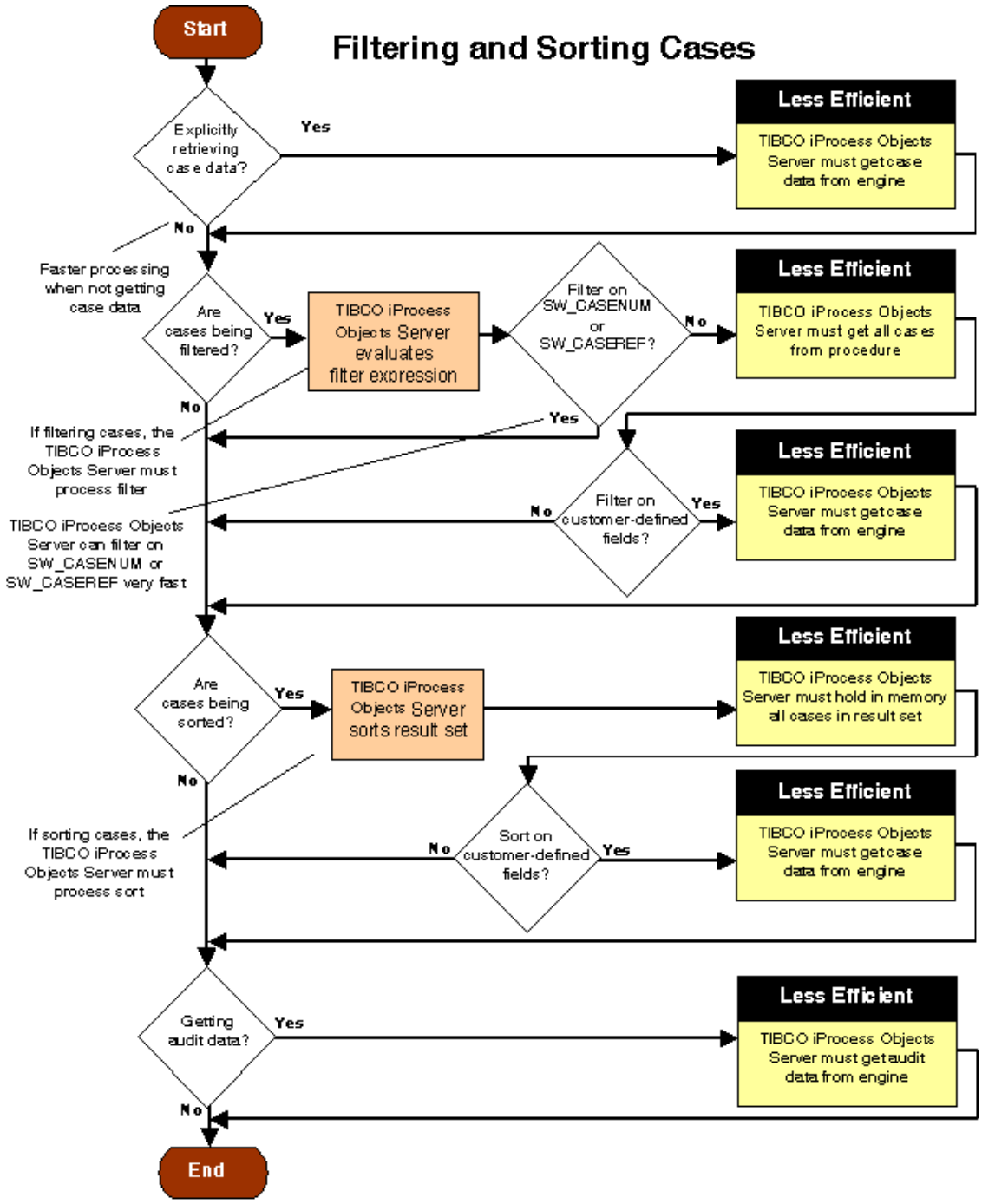
See "Sorting Work Items and Cases" on page 273 for information about setting up sort criteria.

Filtering/Sorting Cases

When filtering and sorting cases:

- Cases are *always* filtered by the TIBCO iProcess Objects Server. To filter cases, the TIBCO iProcess Objects Server must retrieve all cases (both active and closed) from the procedure to be able to filter them. This can take a significant amount of time, depending on the number of cases. The TIBCO iProcess Objects Server can, however, efficiently filter on case number (SW_CASENUM) or case reference number (SW_CASEREF) (see "Efficiently Filtering Cases on the TIBCO iProcess Objects Server" on page 205 for more information). The elements you are allowed to use in your filter expressions to filter cases are listed below.
- If you "get case data" in your application, this causes the filter processing to be less efficient. More about "getting case data" is explained below.
- Cases are *always* sorted by the TIBCO iProcess Objects Server. This, however, requires that the server hold in memory all of the cases in the result set.

The following flow diagram shows the decision process that takes place when filtering and sorting cases:



As shown in the illustration, there are some actions that will cause the filter/sort operation to be less efficient when filtering and sorting cases:

- Getting case data
- Performing the filter operation on the TIBCO iProcess Objects Server
- Performing the sort operation on the TIBCO iProcess Objects Server
- Getting audit data

Additional information about these actions is provided in the subsections that follow.

Getting Case Data

Causing the server to "get case data" means that the TIBCO iProcess Objects Server must retrieve case data from the TIBCO iProcess Engine. This significantly slows down the filter/sort processing. The following actions cause the TIBCO iProcess Objects Server to get case data:

- Explicitly Retrieving Case Data - If you explicitly ask for case data using the *aCaseFieldNames* parameter when constructing a case "content" object (**vACaseContent**), the server will explicitly retrieve the data in those fields from the engine. See "Retrieving Field Data from the Server" on page 167 for information about the use of the *aCaseFieldNames* parameter.
- Having the TIBCO iProcess Objects Server filter on customer-defined fields - The TIBCO iProcess Objects Server does not have direct access to case data. Therefore, if your filter expression contains a customer-defined field (i.e., any field on a form that is not a system field (SW_PRIORITY, SW_PRONAME, etc.)), it must retrieve the data in that field from the TIBCO iProcess Engine, adversely affecting performance.
- Having the TIBCO iProcess Objects Server sort on customer-defined fields - The TIBCO iProcess Objects Server does not have direct access to case data. Therefore, if you sort on a customer-defined field (i.e., any field on a form that is not a system field (SW_PRIORITY, SW_PRONAME, etc.)), it must retrieve the data in that field from the TIBCO iProcess Engine, adversely affecting performance.

Note that although the flow diagram shows that there are three different places where you can take a performance hit by getting case data, the actual hit only occurs once, i.e., you don't take two performance hits, for instance, if you filter on customer-defined fields and sort on customer-defined fields; the TIBCO iProcess Objects Server only has to get case data once for the entire operation.

Filtering Cases on the TIBCO iProcess Objects Server

Cases can be filtered *only* by the TIBCO iProcess Objects Server. This limits your options to perform an efficient filter operation because the TIBCO iProcess Objects Server must always retrieve all cases (both active and closed) from the engine to be able to determine if they satisfy the filter expression. For large numbers of cases this can take a significant amount of time.

The following table lists the elements that can be used in filter expressions when filtering cases:

Element	Description
Logical Operators	AND, OR
Comparison Operators	=, <, >, <=, >=, <> (The ? character can also be used as an equality operator with regular expressions — see "Using Regular Expressions" on page 215.)

Element	Description
System Fields	All system fields that are applicable to cases (see the *Applies To* column in the table of system fields used for filtering — page 208)
Case Data Fields	Case data fields can be included in your filter expressions, although it causes you to take a performance hit because the TIBCO iProcess Objects Server must get case data from the engine.
Wildcards	Note that the '*' and '?' characters are NOT interpreted as wildcard characters when filtering cases on the TIBCO iProcess Objects Server. They are interpreted literally, i.e., as an asterisk and question mark. (This applies when using the '=' equality operator. You can use '*' and '?' as wildcard characters when using the '? equality operator (i.e., with regular expressions — see below).)
Regular Expressions	Regular expressions can be used when filtering cases, allowing you to do complex pattern matching. See "Using Regular Expressions" on page 215.

The following is an example of a filter expression for filtering cases:

- To define a filter expression for all cases that were started on or before March 1, 2003 (assume mm/dd/yyyy date locale setting in the engine), set the filter expression to:

```
"SW_STARTEDDATE <= !03/01/2003!"
```

Efficiently Filtering Cases on the TIBCO iProcess Objects Server

The *Filtering and Sorting Cases* flow diagram shows that if you are filtering cases, you can bypass the performance hit normally caused by filtering on the TIBCO iProcess Objects Server by filtering on either SW_CASENUM or SW_CASEREF.

Cases are indexed by case number (SW_CASENUM) and case reference number (SW_CASEREF). Therefore, if your filter expression contains one (and only one) of these system fields, the TIBCO iProcess Objects Server is able to perform the filtering operation very quickly. When using these system fields, the server does not have to retrieve all of the cases from the procedure.

The following are example filter expression strings using the case number and case reference number:

```
"SW_CASENUM = 150"

"SW_CASEREF = \"2-6\""
```

Note - Case number is an integer; case reference number is a text string.

This exception for cases does not allow for any compound expressions; you can only filter on a single case number or a single case reference number.

Sorting Cases on the TIBCO iProcess Objects Server

As described earlier and shown in the *Filtering and Sorting Cases* illustration, cases are always sorted by the TIBCO iProcess Objects Server. This is not real efficient because the TIBCO iProcess Objects Server **must hold in memory all work items in the filter result set**. If the result set from the filter operation is very large, this can consume a significant amount of memory.

The table below shows the sort criteria you can use when sorting cases.

Sort Criteria for Sorting Cases
• All system fields that are applicable to cases (see the *Applies To* column in the table of system fields used for sorting — page 275. • Case data fields can be included in your sort criteria, although it causes you to take a performance hit because the TIBCO iProcess Objects Server must get case data from the engine.

See "Sorting Work Items and Cases" on page 273 for specific information about setting up sort criteria.

Getting Audit Data

If audit data is requested on the cases in the pageable list, this causes the TIBCO iProcess Objects Server to retrieve the audit data from the engine for each case on which it's requested. This impacts the performance of a case filter operation. (See "Getting Audit Step Objects" on page 72 for information about how audit data is requested.) For efficiency reasons, only include audit data in cases in which it is really needed.

Filter Criteria Format

The following shows the valid format for your filter criteria expressions. This is a BNF-like description. A vertical line "|" indicates alternatives, and [brackets] indicate optional parts.

<criteria>

<exp> | <exp> <logical_op> <exp> | [<criteria>]

<exp>

<value> <comparison_op> <value>

<logical_op>

and | or

<value>

<field> | <constant> | <systemfield>

<comparison_op>

= | <> | ? | < | > | <= | >=

<field>

<alpha>[fieldchars]

<systemfield>

See "System Fields Used in Filtering" on page 208 for a list of the allowable system fields.

<constant>

<date> | <time> | <numeric> | <string>

<date>

!<localdate>!

<time>

#<hour>:<min>#

<datetime>

"<localdate> <hour>:<min>"

<hour>

00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23

<min>

00| 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59

<localdate>

<mm>/<dd>/<yyyy> | <dd>/<mm>/<yyyy> | <yyyy>/<mm>/<dd> | <yyyy>/<dd>/<mm>

<mm>

01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12

<dd>

01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31

Note - The day and month portion of a date must be two digits. Correct: 09/05/2000. Incorrect: 9/5/2000.

<yyyy>

<digit> <digit> <digit> <digit>

<numeric>

<digits> [.<digits>]

<string>

"<asciichars>"

<asciichars>

<asciichar> [<asciichars>]

<asciichar>

ascii characters between values 32 and 126

<alpha>

a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<digit>

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<digits>

<digit> [<digits>]

<alphanum>

<alpha> | <digit>

<alphanums>

<alphanum> [<alphanums>]

<fieldchar>

<alpha> | <digit> | _

<fieldchars>

<fieldchar> [<fieldchars>]

System Fields Used in Filtering

System fields are symbolic references to data about a work item or case. These fields are primarily used by the TIBCO iProcess Engine (specifically, the Work Item Server) when performing filtering and sorting functions. The information that is available to the engine through the system fields is also available to the application though methods on work item and case objects. For example, **SW_CASENUM** is available to the client with the **vCaseId.getCaseNumber** method. The TIBCO iProcess Engine, however, doesn't have access to those methods, so the method names can't be used in filter and sort criteria — instead, the system field names need to be used in your expressions. For example:

```
"SW_CASENUM=5"
```

The system fields that are available for filtering are listed in the table below. Note that some system fields are only applicable for filtering on work items, some only for filtering on cases, and some are applicable to both (see the "Applies to" columns). The WIS-compatible column tells you if the system field is Work Item Server-compatible — see "Can the WIS Perform the Filter Operation?" on page 199 for more information.

Filter Criteria	System Field	Data Type	Length	WIS-compatible	Applies to work items	Applies to cases
Addressee of work item (username@node)	SW_ADDRESSEE	Text	49		X	
Arrival date and time	SW_ARRIVAL	DateTime	16	X	X	
Arrival date	SW_ARRIVALDATE	Date	10	X	X	
Arrival time	SW_ARRIVALTIME	Time	5	X	X	
Case description	SW_CASEDESC	Text	24	X	X	X
Case ID in procedure	SW_CASEID	Numeric	7		X	
Case number	SW_CASENUM	Numeric	15	X	X	X
Case reference number	SW_CASEREF	Text	20	X	X	X
Date (current)	SW_DATE	Date	10		X	X
Deadline date and time	SW_DEADLINE	DateTime	16	X	X	
Deadline date	SW_DEADLINEDATE	Date	10		X	
Deadline expired flag (1 - expired; 0 - all work items)	SW_EXPIRED	Numeric	1	X	X	

Filter Criteria	System Field	Data Type	Length	WIS-compatible	Applies to work items	Applies to cases
Deadline set flag (1 - has deadline; 0 - all work items)	SW_HASDEADLINE	Numeric	1	X	X	
Deadline time	SW_DEADLINETIME	Time	5		X	
Forwardable work item flag (1 - forwardable; 0 - all work items)	SW_FWDABLE	Numeric	1	X	X	
Host name	SW_HOSTNAME	Text	24 or 8[a]	X	X	X
Locker of the work item (username)	SW_LOCKER	Text	24 or 8[a]		X	
Mail ID	SW_MAILID	String or Numeric [b]	45 (String) 7 (Numeric)		X	
Outstanding work item count (not available on TIBCO iPro-cess Engines)	SW_OUTSTANDCNT	Numeric	7			X
Pack file (not available on TIBCO iProcess Engines)	SW_PACKFILE	Text	13		X	
Priority of work item	SW_PRIORITY	Numeric	3	X	X	
Procedure description	SW_PRODESC	Text	24	X	X	X
Procedure name	SW_PRONAME	Text	8	X	X	X
Procedure number	SW_PRONUM	Numeric	7		X	X
Releasable work item (no input fields) (1 - releasable; 0 - all work items)	SW_RELABLE	Numeric	1	X	X	
Started date and time of the case	SW_STARTED	DateTime	16			X
Started date of the case	SW_STARTEDDATE	Date	10			X
Started time of the case	SW_STARTEDTIME	Time	5			X
Starter of the case (username@node)	SW_STARTER	Text	24 or 8[a]		X	X
Status of the case (“A” - active; “C” - closed)	SW_STATUS	Text	1			X
Step (work item) description	SW_STEPDESC	Text	24	X	X	
Step (work item) name	SW_STEPNAME	Text	8	X	X	
Step (work item) number in procedure	SW_STEPNUM	Numeric	7		X	
Suspended work item (1 - suspended; 0 - not suspended) (only available on TIBCO iProcess Engines)	SW_SUSPENDED	Numeric	1		X	

Filter Criteria	System Field	Data Type	Length	WIS-compatible	Applies to work items	Applies to cases
Terminated date and time of the case	SW_TERMINATED [c]	DateTime	16			X
Terminated date of the case	SW_TERMINATEDDATE [c]	Date	10			X
Terminated time of the case	SW_TERMINATEDTIME [c]	Time	5			X
Time (current)	SW_TIME	Time	5		X	X
Unopened work item (1 - unopened; 0 - all work items)	SW_NEW	Numeric	1	X	X	
Urgent flag (1- urgent; 0 - all work items)	SW_URGENT	Numeric	1	X	X	
Work queue parameter 1	SW_QPARAM1	Text	24	X	X	
Work queue parameter 2	SW_QPARAM2	Text	24	X	X	
Work queue parameter 3	SW_QPARAM3	Text	12	X	X	
Work queue parameter 4	SW_QPARAM4	Text	12	X	X	

a. This has a length of 24 for long-name systems, or 8 for short-name systems.

b. If using a TIBCO Process Engine, SW_MAILID is an integer of length 7; if using a TIBCO iProcess Engine, SW_MAILID is a string of length 45.

c. Only cases that have been terminated will be returned when filtering on these system fields. For instance, if your filter expression asks for cases where SW_TERMINATEDDATE < !09/01/2002!, only those cases that ARE terminated and whose termination date is earlier than 09/01/2002 are returned.

Note - The System Fields that can be set to 1 and 0 work in the following manner: When set to 1, only the respective work items are displayed; when set to 0, all work items are displayed. For example, if SW_FWDABLE is set to 1, this means "display only the forwardable work items". If it's set to 0, this means "don't display only the forwardable work items, instead, display all of them."

Data Types used in Filter Criteria

The following are definitions of the different data types used in filter criteria (see the Data Type column in the System Fields table in the previous section).

Data Type	Description
Numeric	Numeric numbers are simply entered in the expression. Examples: 36 or 425.00
Text	Text must be enclosed within double quotes. Example: "Smith"
Date	Dates must be enclosed in exclamation marks. The ordering of the day, month and year is specified in the **staffpms** file (see "Date Format" on page 178). Example: !12/25/1997!
Time	Times can be included in the expression in the format hh:mm. They must be enclosed in pound signs. Uses the 24-hour clock. Example: #18:30#
DateTime	DateTime constants are a combination of a date and time, separated by a space, all enclosed in double quotes. The ordering of the day, month and year is specified in the **staffpms** file (see "Date Format" on page 178). Example: "12/25/1997 10:30"

Note - The day and month portion of a date must be two digits (correct: 09/05/2004; incorrect: 9/5/2004). The year portion of a date must be four digits (correct: 09/05/2004; incorrect: 09/05/04).

Data Type Conversions

If you specify a filter expression that compares values of different types, both types will be converted to strings and compared as strings. For example, assume NUM_FIELD is a TIBCO field of type numeric with a value of 275. The filter:

```
NUM_FIELD < "34"
```

will result in being True because NUM_FIELD will be converted to a string before the comparison is made ("275" < "34").

The expression:

```
!06/03/1999! < 34
```

will be converted to:

```
"06/03/1999" < "34"
```

Filtering on Case Data Fields

You can filter work items in a work queue based on the values in the fields of the work item (referred to as "case data" fields).

There are two ways in which you can filter on case data:

- Using Case Data Queue Parameter (CDQP) Fields - CDQP fields are a more recent addition than Work Queue Parameter fields (see below) that allow you to filter and/or sort on an unlimited number of case data fields that appear in work items on your work queue.
- Using Work Queue Parameter Fields - These fields are used by assigning a case data field value to one of the pre-defined work queue parameter fields, then using the Work Queue Parameter field in filter or sort criteria. These fields have been superseded by CDQP fields as they were considered too limiting since there are only four of them.

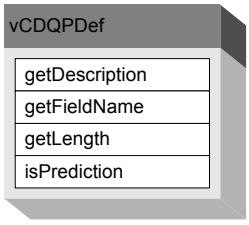
More about CDQP and work queue parameter fields are described in the following subsections.

Using Case Data Queue Parameter Fields

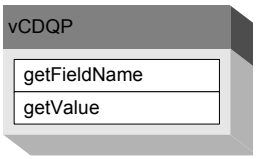
Case Data Queue Parameter (CDQP) fields provide an efficient method of filtering on the value of fields in your work items. To make use of this functionality, you must first pre-designate the fields you want to filter on as CDQP fields. Fields are designated as CDQP fields with the utility, **swutil**. This utility is used to create a list, on the TIBCO iProcess Engine, of the case data fields that are available to use for filtering. For information about using **swutil**, see the *TIBCO iProcess swutil and swbatch Reference Guide*.

Note - Case Data Queue Parameter fields are also used for efficiently sorting on case data, as described in the Sorting Work Items and Cases chapter.

Once you have created the list of CDQP fields with **swutil**, this list of fields is available with the **getCDQPDefs** method. This method, available on both the **sWorkQManager** and **vAWorkQ** objects, returns **vCDQPDef** objects, which provide the CDQP definitions. If called from **sWorkQManager**, it sends a message to the server to retrieve an array of **vCDQPDef** objects, one for each CDQP field defined on the specified work queue. If called from **vAWorkQ**, this returns an array of **vCDQPDef** objects, one for each CDQP field defined on the work queue represented by the local Value Object.



Once you have retrieved a work item Value Object (**vWorkItem**), the CDQP fields that are being used in that work item are available with the **getCDQPs** method. This method returns an array of **vCDQP** objects, one for each CDQP field that is being used in the work item. The **vCDQP** objects provide access to the values in the CDQP fields.



Your filter expressions can include any of the CDQP fields that have been defined on the work queue. For example, assuming LOAN_AMT is listed as one of the CDQP fields for the work queue, the following is a valid filter expression:

```
"LOAN_AMT = 500000"
```

Type of Data in CDQPs

If the WIS is performing the filter or sort operation, and you are using CDQP fields in your filter expression or sort criteria, the evaluation is performed using the “Work Item Data” in the CDQP. Work Item Data reflects “keeps” that have been done on the work item.

If the TIBCO iProcess Objects Server is performing the filter or sort operation, and you are using CDQP fields in your filter expression or sort criteria, the server may perform the evaluation using either “Work Item Data” or “Case Data”, depending on whether or not your TIBCO iProcess Objects Server has implemented CR 12425. If CR 12425 has been implemented in your server, it will evaluate Work Item Data; if CR 12425 has not been implemented in your server, it will evaluate Case Data. Work Item Data reflects “keeps” that have been performed on the work item; Case Data does not reflect “keeps”. (See your TIBCO iProcess Objects Server release notes to determine if CR 12425 is implemented in your server.)

See “Case Data vs. Work Item Data” on page 166 for more information about the difference between Work Item Data and Case Data.

Using Work Queue Parameter Fields

Note - Previous versions of the TIBCO iProcess Objects Server provided “Work Queue Parameter” fields that could be used for filtering and sorting work items based on the value of case data. Work Queue Parameter fields, however, did not provide the flexibility required by some customers. Therefore, a new method of filtering on case data fields has been implemented using “Case Data Queue Parameter” fields (see the previous section). New development should use Case Data Queue Parameter fields instead of the Work Queue Parameter fields (Work Queue Parameter fields will continue to be supported, however).

"Work Queue Parameter" fields allow you to filter work items based on the value of case data fields in your client application. (Work Queue Parameter fields are also used for sorting on case data — see the *Sorting Work Items and Cases* chapter.)

If you have case/field data that you want to filter on (e.g., customer name, loan amount, etc.), it is much more efficient to assign the field value to one of the Work Queue Parameter fields, then filter on that field, instead of directly filtering on the application field. There are four work queue parameter fields available. The default definitions (which can be changed) for these fields are shown below:

Name	Type	Length	Description
SW_QPARAM1	Text	24	WQ Parameter Field 1
SW_QPARAM2	Text	24	WQ Parameter Field 2
SW_QPARAM3	Text	12	WQ Parameter Field 3
SW_QPARAM4	Text	12	WQ Parameter Field 4

These fields can be placed directly in forms, or you can assign the value of an application field to one of the work queue parameter fields through a script. For example:

```
SW_QPARAM1:=LAST_NAME
```

Then, you can filter on the value in the SW_QPARAM1 field. For example, to return only the work items that have a customer last name starting with 'A' through 'M', the filter expression can be set as follows:

```
"SW_QPARAM1?\"[a-m]*\""
```

This would be much more efficient than filtering on the LAST_NAME field.

The **vWorkItem** object contains methods that provide access to the values in the Work Queue Parameter fields — they are **getWorkQParam1 - getWorkQParam4**. These methods return the values you place in the system fields, SW_QPARAM1 - SW_QPARAM4, for each work item.

The **vWorkQ** object contains four methods that return a name for each of the Work Queue Parameter fields — they are **getWorkQParam1Name - getWorkQParam4Name**. If you use the TIBCO iProcess Workspace (Windows), these names appear in the column headers if you display the Work Queue Parameter fields in the Work Queue Manager. For information about modifying these names, see the *TIBCO iProcess Workspace (Windows) Manager's Guide*.

Work Queue Parameter Fields vs. Case Data Queue Parameter Fields

Why would you want to use the new Case Data Queue Parameter (CDQP) fields instead of the older Work Queue Parameter fields? The reasons for using each method is shown in the following table.

Case Data Filtering Method	Reasons For Using This Type
Work Queue Parameter Fields	• They are pre-configured, not requiring any administration (where as, CDQP fields require some additional administration). • They are available for all queues, requiring no additional administration. • They are already taking up resources (memory and disk space) whether they are used or not. (Adding four CDQP fields instead of using the already available Work Queue Parameter fields takes up additional resources.) • The load on the Work Item Server is slightly increased for each CDQP. • Configuring CDQP fields requires a TIBCO iProcess Engine shutdown.
Case Data Queue Parameter Fields	The primary reason to use CDQP fields is because if you use the four available Work Queue Parameter fields, then later realize you need more, it will require application changes — with CDQPs, you can just keep adding as many as needed.

Using Regular Expressions

Regular expressions may be included in filter expressions to provide powerful text search capabilities. They can be used when filtering either work items or cases.

Regular expressions must be in the following format:

constant ? "regular expression"

where:

- **constant** - A constant value or field name. If a field name is included in the expression, the field must be defined as a text data type (SWFieldType = swText). (Note that although the value in DateTime fields (e.g., SW_STARTED) is enclosed in quotes, they cannot be used with regular expressions, as they are not of text data type.)
- **?** - Special character signifying that a regular expression follows (interpreted as an equality operator).
- "**regular expression**" - Any valid regular expression (enclosed in double quotes).

A regular expression (RE) specifies a set of character strings. A member of this set of strings is "matched" by the RE.

The following one-character REs match a single character.

1. An ordinary character (not one of those discussed in number 2 below) is a one-character RE that matches itself. For example, an RE of “a” will match all constants/fields that match “a” exactly.
2. A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

   *, ?, [, and \ Asterisk, question mark, left square bracket, and backslash, respectively. These are always special, except when they appear within square brackets ([ ]; see Item 5 below).

3. An asterisk (*) is a one-character RE that matches zero or more of any character.
4. A question mark (?) is a one-character RE that matches any character except new-line.
5. A non-empty string of characters enclosed in square brackets ([ ]) is a one-character RE that matches any one character in that string, with these additional rules:
   - If the first character of the string is a circumflex (^), the one-character RE matches any character except new-line and the remaining characters in the string. The ^ has this special meaning only if it occurs first in the string.
   - The minus (-) may be used to indicate a range of consecutive characters. For example, [0-9] is equivalent to [0123456789]. The minus sign loses this special meaning if it occurs first (after an initial ^, if any) or last in the string.
   - The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any). For example, [ ]a-f] matches either a right square bracket (]) or one of the ASCII letters a through f, inclusive.
   - The special characters *, ?, [, and \ stand for themselves within such a string of characters.

The following rules may be used to construct REs from one-character REs:

1. A one-character RE is an RE that matches whatever the one-character RE matches.
2. The concatenation of REs is an RE that matches the concatenation of the strings matched by each component of the RE. For example, an RE of “abc” will match all constants/fields that contain “abc” exactly.

Using Escape Characters in the Filter Expression

Filter expressions require a string value. Therefore, if within the string value, you are required to provide another string, you must use an escape character to provide the quoted string within a string.

Use the back slash to indicate that the next character is a special character. In the example below, the back slashes indicate that the quotes that follow them are quoting the string "LOAN", and are not the ending quotes for the filter expression string.

```
"SW_PRONAME=\"LOAN\""
```

Filtering on Empty Fields

To filter on an empty field, compare the field with **SW_NA**, which checks to see if the field is "not assigned." For example:

```
"SOC_SEC_NUM=SW_NA"
```

This returns only work items in which the SOC_SEC_NUM field is empty.

Comparing the field with an empty set of quotes (SOC_SEC_NUM="") will cause all work items to be returned. Here's why: The TIBCO iProcess Objects Server determines that this is a filter that the Work Item Server can perform. The Work Item Server views the filter from the perspective of the Work Queue Manager. When you set up filter criteria from within the Work Queue Manager, if you leave a filter field blank, it means "match all items." That's essentially what you are doing when you compare one of the Work Queue Manager-defined filter criteria to an empty set of quotes.

How to Specify Ranges of Values

Ranges of values can be specified in your filter expressions. This functionality, however, is limited to filtering on work items only — you cannot use range filtering when filtering cases.

Ranges must use the following format:

```
FilterField=[val1-val2|val3|val4-val5|.....|valn]
```

You can specify multiple ranges or single values, each separated by a vertical bar. The entire range expression is enclosed in square brackets. Only the '=' equality operator is allowed in a range filter expression.

Dates are specified as:

```
!dd/mm/yyyy!
```

Note - The ordering of the day, month and year is specified in the ***staffpms*** *file (see* *"Date Format" on page 178**).*

Times are specified as:

```
#mm:hh#
```

DateTimes are specified as:

```
"dd/mm/yyyy mm:hh"
```

Range Filter Example 1:

This example returns the work items with case numbers between 50 and 100, and between 125 and 150, as well as the work item with case number 110:

```
SW_CASENUM=[50-100|110|125-150]
```

Range Filter Example 2:

To return all work items that arrived in the queue between 09/01/2000 and 09/03/2000 (inclusive), and that have a priority equal to 50:

```
SW_ARRIVALDATE=[!09/01/2000! - !09/03/2000!] AND SW_PRIORITY=50
```

Specifying Multiple Ranges

When setting up a range filter, if you are filtering on criteria that can be filtered through the Work Queue Manager, you are limited to five ranges in the expression if you want the Work Item Server to evaluate the expression (which is what you want — the Work Item Server processes filter expressions much faster than the TIBCO iProcess Objects Server). For instance, if you are filtering on the case number, you can specify up to five case number ranges in your filter expression:

```
SW_CASENUM=[50-100|110|125-150|180-200|225-250]
```

The reason for this is because the TIBCO iProcess Objects Server always first determines if the filter expression is something the Work Item Server can handle. If it is, the TIBCO iProcess Objects Server sends it to the Work Item Server to evaluate the filter expression; otherwise the TIBCO iProcess Objects Server will evaluate it. When the filter assignment is sent to the Work Item Server, you must

abide by the rules/limitations of filtering through the Work Queue Manager. One of the limitations is that when defining filter ranges in the Work Queue Manager, there are only five range filter fields in which you can enter filter criteria.

If you exceed five ranges in your filter expression, the TIBCO iProcess Objects Server must evaluate the expression, which is a lot less efficient than the Work Item Server.

The following is a list of the filter criteria for which you can enter ranges in the Work Queue Manager. Each of these is limited to five separate ranges:

- Host Name (SW_HOSTNAME)
- Procedure Name (SW_PRONAME)
- Procedure Description (SW_PRODESC)
- Case Number (SW_CASENUM)
- Case Description (SW_CASEDESC)
- Form Name (SW_STEPNAME)
- Form Description (SW_STEPDESC)
- Deadline (SW_DEADLINE)
- Priority (SW_PRIORITY)
- WQ Parameter 1 (SW_QPARAM1)
- WQ Parameter 2 (SW_QPARAM2)
- WQ Parameter 3 (SW_QPARAM3)
- WQ Parameter 4 (SW_QPARAM4)

Closing/Purging Cases Based on Filter Criteria

The **sCaseManager** object contains methods that allow you to close or purge cases based on filter criteria. These methods are:

- **closeCasesByCriteria** - This method closes cases that match the specified filter criteria. You must have case administration privilege to close a case (defined in TIBCO Business Studio). You also cannot close a case from a slave node.
- **purgeCasesByCriteria** - This method purges cases that match the specified filter criteria. You must have case administration privilege to purge a case (defined in TIBCO Business Studio). You also cannot purge a case from a slave node.

Both of these methods require a parameter that specifies a filter string expression. Use the filter expression syntax described in this chapter.

Closing and purging cases require that the user have system administrator authority (MENUNAME = ADMIN). See "User Attributes" on page 187 for information about the MENUNAME attribute.

Default Filter Criteria

The TIBCO iProcess Server Objects provide the ability to set *default* work item filter and sort criteria for a work queue.

Default criteria is a feature of the TIBCO iProcess Objects Server that allows you to save a specific criteria that persists for the work queue. Note, however, that the default criteria is not automatically applied to a pageable list of work items that is created for that work queue. Filter criteria must *always* be passed in the form of a **vWICriteria** Value Object when the list is requested. To apply the default criteria, you must call the **getDefaultCriteria** method (which returns a **vWICriteria** object representing the default criteria), then pass the **vWICriteria** object to the **getWorkItemList** or **getAWorkItemList** method when requesting the pageable list.

Note - If you use the TIBCO iProcess Workspace (Windows), filter criteria that are defined on the Work Queue Manager Work Item List Filter dialog become the default filter criteria for that work queue. The default filter criteria defined on this dialog can be viewed and/or affected by the methods described below.

The **sWorkQ** object contains methods that allow you to affect the default filter criteria (note that at the same time these methods are affecting the default sort criteria for the work queue — see "Setting Default Sort Criteria" on page 279):

- **changeDefaultCriteria** - This method sets the default criteria for the work queue based on the **vWICriteria** object passed in the method call. These filter criteria will persist on this work queue until changed again with this method or cleared with the **clearDefaultCriteria** method. (Note that this method is also setting the default sort criteria based on sort fields that are included in the **vWICriteria** object passed in the method call.)
- **clearDefaultCriteria** - This method clears the default filter criteria that were set either through the Work Queue Manager or by using the **changeDefaultCriteria** method. (Note that this also clears any default sort criteria that have been defined.)
- **getDefaultCriteria** - This method returns a **vWICriteria** object, indicating the currently set default criteria for the work queue. To apply default criteria, you must call this method to obtain the **vWICriteria** object, then pass that Value Object with the **getWorkItemList** or **getAWorkItemList** method when requesting the pageable list of work items.

You can **only** persist filter criteria that are a subset of those supported by the Work Queue Manager or an exception will be thrown when you execute **changeDefaultCriteria**. The following are the filter criteria that are supported by the Work Queue Manager.

System Field	Description
SW_ARRIVAL	Arrival date and time
SW_ARRIVALTIME	Arrival time
SW_ARRIVALDATE	Arrival date
SW_CASEDESC	Case description
SW_CASENUM	Case number
SW_CASEREF	Case reference number
SW_DEADLINE	Deadline date and time
SW_DEADLINETIME	Deadline time
SW_DEADLINEDATE	Deadline date

System Field	Description
SW_EXPIRED	Deadline Expired Flag
SW_FWDABLE	Forwardable Items
SW_HASDEADLINE	Deadline Set Flag
SW_HOSTNAME	Host Name
SW_NEW	Unopened Work Item Flag
SW_PRIORITY	Priority of work item
SW_PRODESC	Procedure Description
SW_PRONAME	Procedure Name
SW_QPARAM1	Work Queue Parameter1
SW_QPARAM2	Work Queue Parameter2
SW_QPARAM3	Work Queue Parameter3
SW_QPARAM4	Work Queue Parameter4
SW_RELABLE	Releasable Work Item Flag
SW_STEPDESC	Form (Step) Description
SW_STEPNAME	Form (Step) Name
SW_URGENT	Urgent Work Item Flag

Also note the when using the **changeDefaultCriteria** method, your filter expressions must conform to the following guidelines:

- The *only* equality operator you can use is ‘=’.
- You cannot use any of the following equality operators: ‘?’,’<‘, ‘<=’, ‘>’, ‘>=’, and ‘<>’.
- You cannot use the OR logical operator.
- And since ‘?’ is not allowed, no regular expression syntax can be used.

14

Filtering Work Items and Cases

With WIS Work Item Filtering

Important - Read this page first to determine which of the *Filtering Work Items and Cases* chapters you should use.

Over time, enhancements have been made to the TIBCO iProcess Objects Server to improve the efficiency of filtering and sorting work items and cases. Because the scope of the enhancements is fairly major, three chapters are now provided in this guide that describe how filtering and sorting work, depending on which of the enhancements have been implemented in your TIBCO iProcess Objects Server. Use the table below to determine which chapter to use, based on the enhancements in your TIBCO iProcess Objects Server.

Note - Although the topic of sorting is covered in a separate chapter, filtering and sorting is described as a single process in the Filtering Work Items and Cases chapters because that is the way it is performed — work items or cases are filtered, then the result set from the filter operation is sorted.

Two major enhancements have been added to the TIBCO iProcess Objects Server that impact filtering and sorting:

- **WIS Work Item Filtering** - This enhancement moved all work item filter processing to the Work Item Server (WIS). With this enhancement, all of the additional capabilities previously provided by the TIBCO iProcess Objects Server can now be performed by the WIS when filtering work items (such as allowing the OR logical operator, allowing the <, >, <=, >=, and <> operators, etc.). Since the WIS has the work items cached, and has direct access to case data, this provides for very efficient filtering and sorting of work items.

  Your server/engine must have the following CRs implemented for this enhancement: TIBCO iProcess Objects Server - CR 12744; TIBCO Process/iProcess Engine - CR 12686.

- **Database Case Filtering** - This enhancement moved all case filter and sort processing to the database. With this enhancement, the filter expression is translated into an SQL select statement, which is used to create the result set from the cases in the database. The result set is then sorted. Because of the indexing ability of the database, this provides for very efficient filtering and sorting of cases.

  This enhancement was implemented in the following CRs: TIBCO iProcess Objects Server - CR 13182; TIBCO Process/iProcess Engine - CR 13098.

Use the following table to determine which of the *Filtering Work Items and Cases* chapters to use:

If your TIBCO iProcess Objects Server includes...	Use this chapter...
Neither of the enhancements listed above	Chapter 13
Only the WIS Work Item Filtering enhancement (CR 12744)	Chapter 14
Both the WIS Work Item Filtering and the Database Case Filtering enhancements (CRs 12744 and 13182)	Chapter 15

Introduction

The TIBCO iProcess Server Objects provide the ability to *filter* work items and cases, allowing you to filter out all those you aren't currently interested in. For example, you may only be interested in the work items that arrived in the work queue today, in which case you could specify a filter expression that filters out all work items other than those that arrived today:

```
"SW_ARRIVALDATE = !08/02/2001!"
```

The benefits of this are two-fold:

- It allows you to display to the user only those cases or work items that are of interest to them.
- It reduces the amount of work the client and the server need to do. When the result set from the filter operation results in fewer work items or cases, this reduces the work load on the client and server.

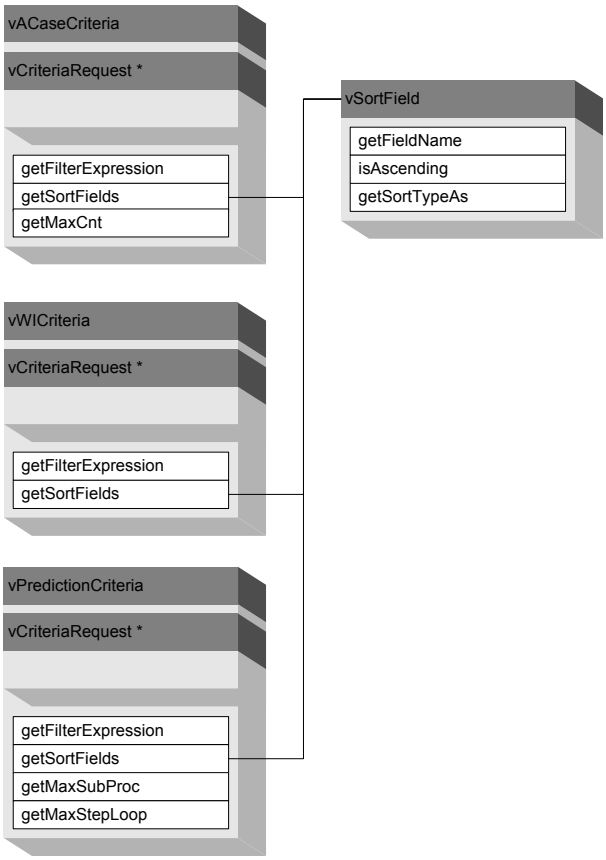
Specifying Filter Criteria

The TIBCO iProcess Server Objects provide three "criteria" objects that specify filter criteria:

- **vACaseCriteria** (for cases)
- **vWICriteria** (for work items)
- **vPredictionCriteria** (for predicted work items) (Note that since predicted items are stored in the database, they are filtered in the same way as cases when you have the database case filtering enhancement — see *Filtering Work Items and Cases* on page 247 for more information.)

Note that these criteria objects are used for both filtering and sorting work items and cases — see "Sorting Work Items and Cases" on page 273 for information about sorting.

Work items and cases that can be filtered are always returned in pageable lists (**sPageableList**, **sPageableListR**, or **sPageableListJ** objects) — see "Working with Lists" on page 137 for information about using pageable lists after the filtered work items or cases have been returned from the server.



The methods that return filtered work items and cases are summarized in the table below.

Method	Uses this Criteria Object	Returns Pageable List of this Object
sWorkQ.getWorkItemList	vWICriteria	vWorkItem
sWorkQ.getAWorkItemList	vWICriteria	vAWorkItem
sCaseManager.getACaseList	vACaseCriteria	vACase
sCaseManager.getPredictedItemList	vPredictionCriteria	vPredictedItem

To specify filter criteria for a pageable list of work items or cases, follow these steps:

1. Construct a **vACaseCriteria**, **vWICriteria**, or **vPredictionCriteria** object, setting the *aFilterExpression* parameter to the desired filter expression string — see the "Defining Filter Expressions" section below for specifics about creating these strings.
2. Pass the **vACaseCriteria**, **vWICriteria**, **vPredictionCriteria** object as an input parameter with one of the methods listed in the table above, depending on the type of object you want returned in the pageable list.

Defining Filter Expressions

To filter work items, cases, or predicted work items, you must define a *filter expression* string, which is passed in the *aFilterExpression* parameter with the applicable criteria object (see the previous subsection). The filter expression string is evaluated against each work item or case you are requesting, returning either True or False. If it returns True, the work item/case is included in the pageable list; if it returns False, the work item/case is not included in the pageable list. For example, you may choose to filter the list to include only work items with the urgent flag set ("SW_URGENT = True").

Filter expression strings can contain elements such as system fields (SW_CASENUM, SW_NEW, etc.), logical operators (AND, OR), comparison operators (=, <, <=, etc.), ranges of values, etc. Details about filter expressions is described in the subsections that follow.

Note that the left and right side of comparison operators (=, <, >, <=, >=, <>, ?) must each consist of only a single field name or single constant. It cannot be an expression containing operators (+, -, /, *, etc.).

Example 1:
To define a filter for work items matching all new items from procedure LOANS, set the filter expression to the following string:

```
"SW_NEW=1 AND SW_PRONAME=\"LOANS\""
```

Example 2:
To define a filter for all work items that arrived after June 20, 2001 (assuming mm/dd/yyyy date locale setting in the engine), set the filter expression to the following string:

```
"SW_ARRIVALDATE > !06/20/2001!"
```

Example 3:

To define a filter for all cases with field LOAN_AMT having a value greater than 100000, set the filter expression to the following string:

```
"LOAN_AMT > 100000"
```

Number of Cases or Work Items in a Filtered Pageable List

There are a number of methods available that provide information about the number of cases or work items on a filtered pageable list. They are:

On **sPageableList**:

- **Available Count (getAvailableCnt) -** This returns the total number of items available in the indexed collection on the TIBCO iProcess Objects Server. This count includes only the work items or cases that satisfy the filter criteria.
- **Local Count (getLocalCnt) -** This returns the number of cases or work items currently being held in the local bock(s) on the sPageableList. If **isKeepLocalItems** has been set to False, this count will always be less than or equal to the number of items per block (**getItemsPerBlock**).

Note - The counts above require an understanding of how an indexed collection is created on the TIBCO iProcess Objects Server, and how blocks of objects are held locally in the pageable list. See *"Working with Lists" on page 137* *for information.*

On **vSummary**:

- **Exclude Count (getExcludeCnt) -** This returns the number of cases or work items that did not satisfy the specified filter criteria, and therefore, were not included in the pageable list.
- **Invalid Count (getInvalidCnt) -** When filtering cases, this returns the number of cases not included in the pageable list because they were invalid within the context of the filter criteria (e.g., the filter expression references a field name not defined in the procedure).
- **Over Maximum Count (getOverMaxCnt) -** This returns the number of cases that were *not* returned from the server because the number returned was limited using the *aMaxCnt* parameter on the **vACaseCriteria** constructor when retrieving a list of cases. This is applicable only when retreiving cases.

  When filtering work items, this method is no longer applicable (it returns -1 if the pageable list contains work items).

On **vWISummary**:

These counts are applicable only if the pageable list contains work items.

- **Urgent Count (getUrgentCnt) -** This returns the number of work items on the pageable list that are marked as urgent. A work item is marked as urgent if its priority value (vWorkItem.getPriorty) is less than or equal to a specific value. By default, the value is 10, although it can be modified in the **staffcfg** file.
- **Deadline Count (getDeadlineCnt) -** This returns the number of work items on the pageable list that have deadlines.
- **Unopened Count (getUnopenedCnt) -** This returns the number of work items on the pageable list that have not been opened (locked).

Filtering/Sorting in an Efficient Manner

The way in which you write your filter expressions can have an affect on how efficiently they are evaluated. This section provides guidelines about what types of elements you can include in your filter expressions (and those you should avoid) to ensure an efficient filter operation.

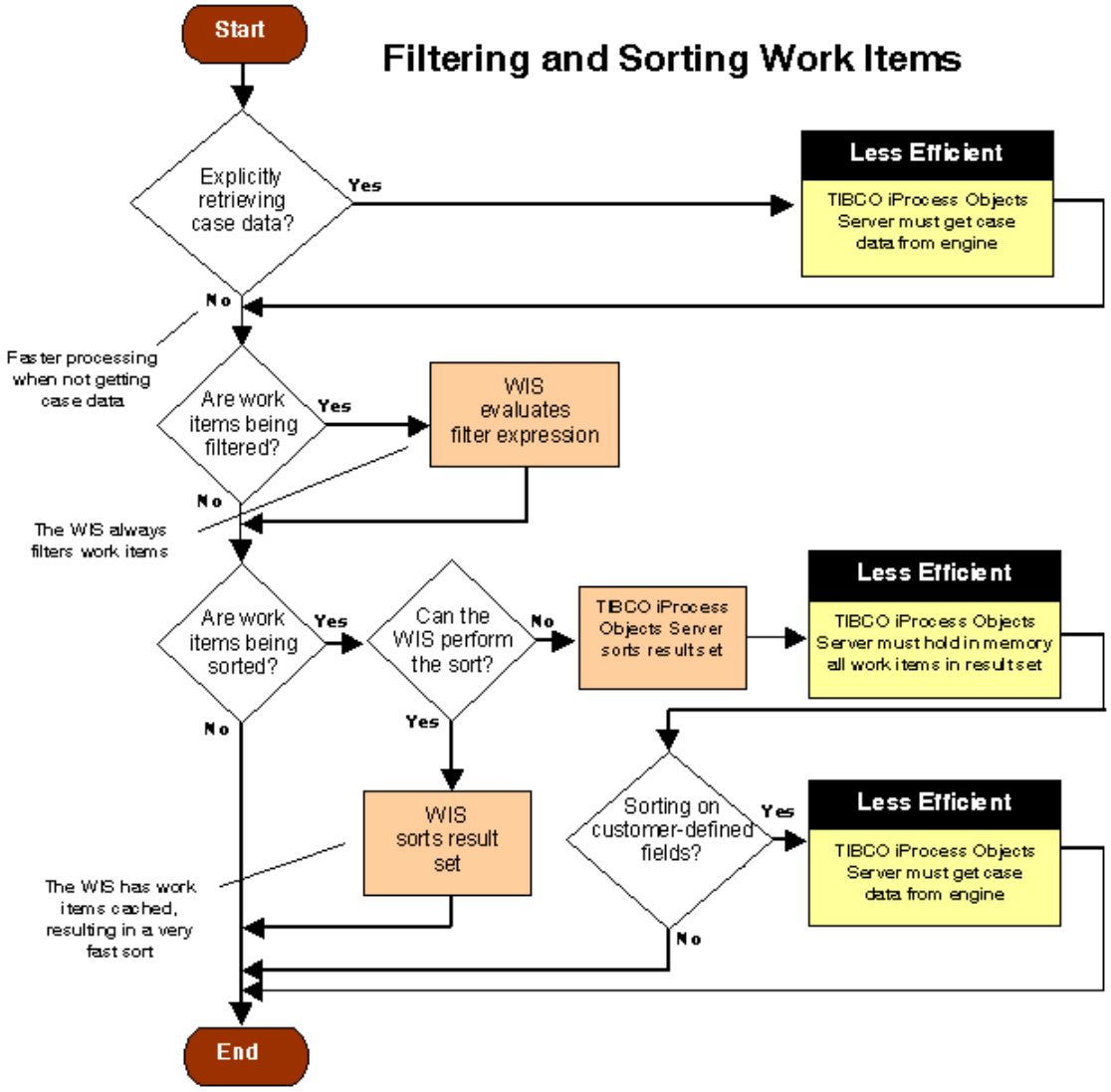
Flow diagrams (one for work items; one for cases) are shown in the following subsections that illustrate the decision process that takes place during a filter/sort operation. Note that the flow diagrams show filtering and sorting taking place in a single operation; that is the way filtering and sorting is processed— works items or cases are filtered to create a result set, then the result set is sorted. The flow diagrams also illustrate how to prevent the filter/sort operation from being less efficient.

Filtering/Sorting Work Items

When filtering and sorting work items:

- Work items are *always* filtered by the Work Item Server (WIS). The WIS has work items cached in memory, allowing it to evaluate filter expressions for work items very quickly. The elements you are allowed to use in your filter expressions are listed in "Work Items are Filtered by the WIS" on page 227.
- If you "get case data" in your application, this causes the filter processing to be less efficient. More about "getting case data" is explained below.
- Work items can be sorted by either the WIS or the TIBCO iProcess Objects Server, depending on how you specify the sort criteria. It's preferable to have the WIS sort the result set from the filter operation. This is explained in detail below.

The following flow diagram shows the decision process that takes place when filtering and sorting work items.



As shown in the illustration, there are a couple of actions that will cause the filter/sort operation to be less efficient when filtering and sorting work items:

- Getting case data
- Performing the sort operation on the TIBCO iProcess Objects Server

Additional information about these actions is provided in the subsections that follow.

Getting Case Data

Causing the server to "get case data" means that the TIBCO iProcess Objects Server must retrieve case data from the TIBCO iProcess Engine. This significantly slows down the filter/sort processing. The following actions cause the TIBCO iProcess Objects Server to get case data:

- Explicitly Retrieving Case Data - If you explicitly ask for case data using the *aCaseFieldNames* parameter when constructing a work item "content" object (**vWIContent**), the server will explicitly retrieve the data in those fields from the engine. See "Retrieving Field Data from the Server" on page 167 for information about the use of the *aCaseFieldNames* parameter.
- Having the TIBCO iProcess Objects Server sort on customer-defined fields - The TIBCO iProcess Objects Server does not have direct access to case data. Therefore, if the sort operation is being handled by the TIBCO iProcess Objects Server, and you sort on a customer-defined field (i.e., any field on a form that is not a system field (SW_PRIORITY, SW_PRONAME, etc.)), the TIBCO iProcess Objects Server must retrieve the data in that field from the engine, adversely affecting performance.

Note that although the flow diagram shows that there are two different places where you can take a performance hit by getting case data, the actual hit only occurs once, i.e., you don't take two performance hits, for instance, if you explicitly retrieve case data *and* the TIBCO iProcess Objects Server is sorting on customer-defined fields; the TIBCO iProcess Objects Server only has to get case data once for the entire operation.

Work Items are Filtered by the WIS

As shown in the *Filtering and Sorting Work Items* illustration, work items are always filtered by the WIS. The WIS has work items cached in memory, allowing it to evaluate filter expressions for work items very quickly.

The following table lists the elements that can be used in filter expressions when filtering work items:

Element	Description
Comparison Operators	=, <, >, <=, >=, <> (The ? character can also be used as an equality operator with regular expressions — see "Using Regular Expressions" on page 241.)
Logical Operators	AND, OR
System Fields	All system fields that are applicable to work items (see the *Applies To* column in the table of system fields used for filtering — page 234)
Case Data Fields	Case data fields can be included in your filter expressions ONLY if they are first defined as CDQPs. If your filter expression references a field that is not a CDQP, the WIS will return a syntax error, which causes the entire filter operation to fail. See "Filtering on Case Data Fields" on page 238 for information.
Wildcards	The wildcard characters '*' and '?' as part of a string on equality checks. The '*' character matches zero or more of any character. The '?' character matches any single character.
Ranges of Values	Ranges of values can be included in your work item filter expressions by using a specific syntax — see "How to Specify Ranges of Values" on page 244 for information.
Regular Expressions	Regular expressions can be used when filtering work items, allowing you to do complex pattern matching. See "Using Regular Expressions" on page 241.

The following is an example of a filter expression for filtering work items:

- To define a filter for all unopened work items, set the filter expression to:

```
"SW_NEW = 1"
```

Can the WIS Perform the Sort Operation?

Work items can be sorted by either the WIS or the TIBCO iProcess Objects Server, depending on the sort criteria you use. Whenever possible, you should use the sort criteria that can be evaluated by the WIS. If the TIBCO iProcess Objects Server must perform the sort operation, **it must hold in memory all work items in the filter result set**. If the result set from the filter operation is very large, this can consume a significant amount of memory.

The table below shows the sort criteria you can use to cause the sort operation to be performed by the WIS. It also lists the expanded criteria available by the TIBCO iProcess Objects Server. Using this expanded criteria causes the sort operation to be performed by the TIBCO iProcess Objects Server, which is less efficient because it must hold the result set in memory.

Sort Criteria the WIS can Process
• System fields that are "WIS-compatible". See the WIS-compatible column in the table of *System Fields used in Sorting* on page 275. (The system fields must be applicable to filtering work items.) • Case Data Queue Parameter (CDQP) fields. See "Sorting on Case Data Fields" on page 277 for more information.
Sort Criteria the TIBCO iProcess Objects Server must Process
• System fields that are NOT "WIS-compatible". See the WIS-compatible column in the table of *System Fields used in Sorting* on page 275. (The system fields must be applicable to filtering work items.) • Case data fields that have NOT been designated as Case Data Queue Parameter (CDQP) fields. See "Sorting on Case Data Fields" on page 277 for more information.

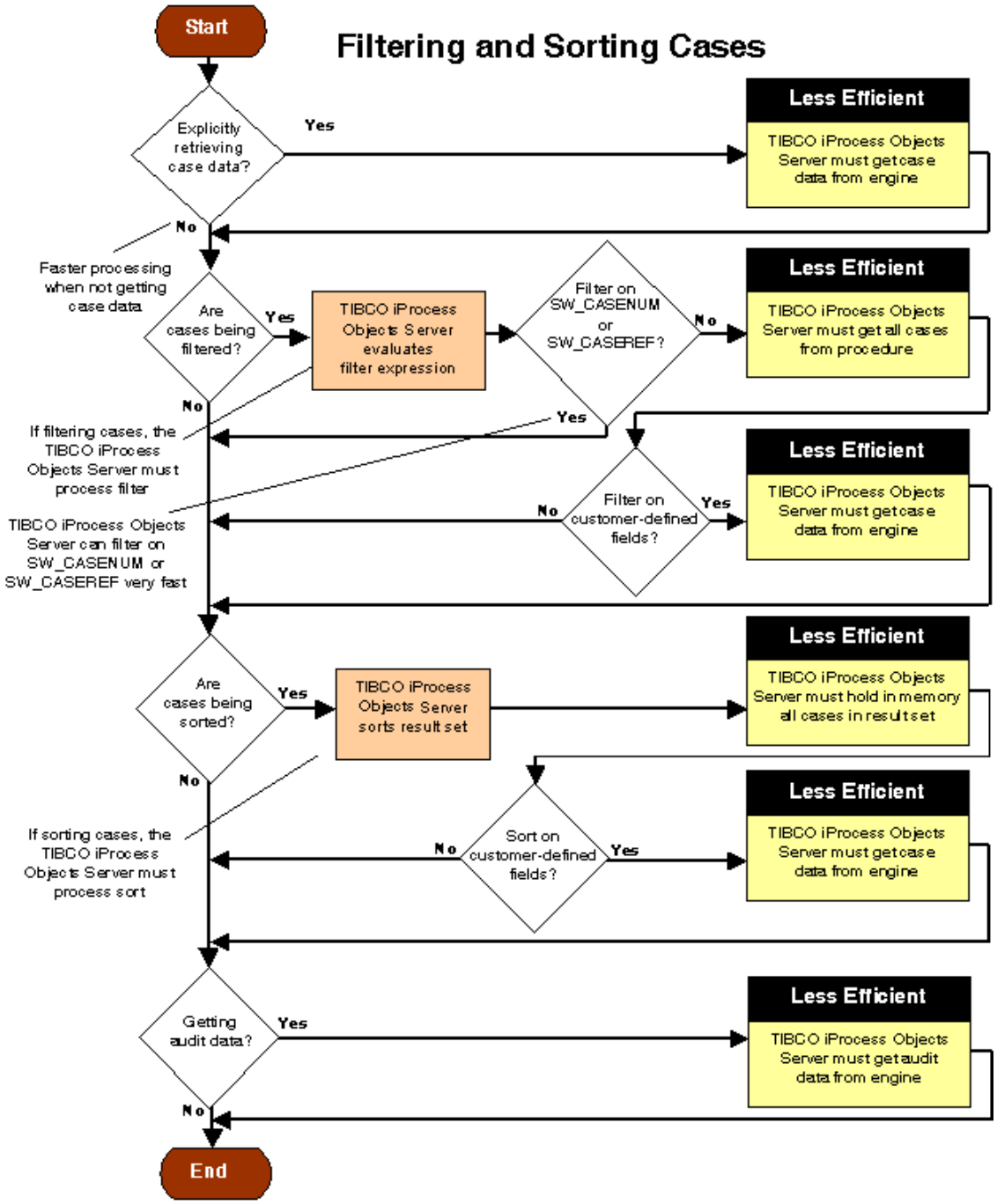
See "Sorting Work Items and Cases" on page 273 for information about setting up sort criteria.

Filtering/Sorting Cases

When filtering and sorting cases:

- Cases are *always* filtered by the TIBCO iProcess Objects Server. To filter cases, the TIBCO iProcess Objects Server must retrieve all cases (both active and closed) from the procedure to be able to filter them. This can take a significant amount of time, depending on the number of cases. The TIBCO iProcess Objects Server can, however, efficiently filter on case number (SW_CASENUM) or case reference number (SW_CASEREF) (see "Efficiently Filtering Cases on the TIBCO iProcess Objects Server" on page 231 for more information). The elements you are allowed to use in your filter expressions to filter cases are listed below.
- If you "get case data" in your application, this causes the filter processing to be less efficient. More about "getting case data" is explained below.
- Cases are *always* sorted by the TIBCO iProcess Objects Server. This, however, requires that the server hold in memory all of the cases in the result set.

The following flow diagram shows the decision process that takes place when filtering and sorting cases.



As shown in the illustration, there are some actions you should avoid, if possible, when filtering and sorting cases:

- Getting case data
- Performing the filter operation on the TIBCO iProcess Objects Server
- Performing the sort operation on the TIBCO iProcess Objects Server
- Getting audit data

Additional information about these actions is provided in the subsections that follow.

Getting Case Data

Causing the server to "get case data" means that the TIBCO iProcess Objects Server must retrieve case data from the TIBCO iProcess Engine. This significantly slows down the filter/sort processing. The following actions cause the TIBCO iProcess Objects Server to get case data:

- Explicitly Retrieving Case Data - If you explicitly ask for case data using the *aCaseFieldNames* parameter when constructing a case "content" object (**vACaseContent**), the server will explicitly retrieve the data in those fields from the engine. See "Retrieving Field Data from the Server" on page 167 for information about the use of the *aCaseFieldNames* parameter.
- Having the TIBCO iProcess Objects Server filter on customer-defined fields - The TIBCO iProcess Objects Server does not have direct access to case data. Therefore, if your filter expression contains a customer-defined field (i.e., any field on a form that is not a system field (SW_PRIORITY, SW_PRONAME, etc.)), it must retrieve the data in that field from the TIBCO iProcess Engine, adversely affecting performance.
- Having the TIBCO iProcess Objects Server sort on customer-defined fields - The TIBCO iProcess Objects Server does not have direct access to case data. Therefore, if you sort on a customer-defined field (i.e., any field on a form that is not a system field (SW_PRIORITY, SW_PRONAME, etc.)), it must retrieve the data in that field from the TIBCO iProcess Engine, adversely affecting performance.

Note that although the flow diagram shows that there are three different places where you can take a performance hit by getting case data, the actual hit only occurs once, i.e., you don't take two performance hits, for instance, if you filter on customer-defined fields and sort on customer-defined fields; the TIBCO iProcess Objects Server only has to get case data once for the entire operation.

The TIBCO iProcess Objects Server Filters Cases

Cases can be filtered *only* by the TIBCO iProcess Objects Server. This limits your options to perform an efficient filter operation because the TIBCO iProcess Objects Server must always retrieve all cases (both active and closed) from the engine to be able to determine if they satisfy the filter expression. For large numbers of cases this can take a significant amount of time.

The following table lists the elements that can be used in filter expressions when filtering cases:

Element	Description
Logical Operators	AND, OR
Comparison Operators	=, <, >, <=, >=, <> (The ? character can also be used as an equality operator with regular expressions — see "Using Regular Expressions" on page 241.)
System Fields	All system fields that are applicable to cases (see the *Applies To* column in the table of system fields used for filtering — page 234)
Case Data Fields	Case data fields can be included in your filter expressions, although it causes you to take a performance hit because the TIBCO iProcess Objects Server must get case data from the engine.

Element	Description
Wildcards	Note that the '*' and '?' characters are NOT interpreted as wildcard characters when filtering cases on the TIBCO iProcess Objects Server. They are interpreted literally, i.e., as an asterisk and question mark. (This applies when using the '=' equality operator. You can use '*' and '?' as wildcard characters when using the '? equality operator (i.e., with regular expressions — see below).)
Regular Expressions	Regular expressions can be used when filtering cases, allowing you to do complex pattern matching. See "Using Regular Expressions" on page 241.

The following is an example of a filter expression for filtering cases:

- To define a filter for all cases that were started on or before March 1, 2003 (assume mm/dd/yyyy date locale setting in the engine), set the filter expression to:

```
"SW_STARTEDDATE <= !03/01/2003!"
```

Efficiently Filtering Cases on the TIBCO iProcess Objects Server

The *Filtering and Sorting Cases* flow diagram shows that if you are filtering cases, you can bypass the performance hit normally caused by filtering on the TIBCO iProcess Objects Server by filtering on either SW_CASENUM or SW_CASEREF.

Cases are indexed by case number (SW_CASENUM) and case reference number (SW_CASEREF). Therefore, if your filter expression contains one (and only one) of these system fields, the TIBCO iProcess Objects Server is able to perform the filtering operation very quickly. When using these system fields, the server does not have to retrieve all of the cases from the procedure.

The following are example filter expression strings using the case number and case reference number:

```
"SW_CASENUM = 150"

"SW_CASEREF = \"2-6\""
```

Note - Case number is an integer; case reference number is a text string.

This exception for cases does not allow for any compound expressions; you can only filter on a single case number or a single case reference number.

The TIBCO iProcess Objects Server Sorts Cases

As described earlier and shown in the *Filtering and Sorting Cases* illustration, cases are always sorted by the TIBCO iProcess Objects Server. This is not real efficient because the TIBCO iProcess Objects Server **must hold in memory all work items in the filter result set**. If the result set from the filter operation is very large, this can consume a significant amount of memory.

The table below shows the sort criteria you can use when sorting cases.

Sort Criteria for Sorting Cases
• All system fields that are applicable to cases (see the *Applies To* column in the table of system fields used for sorting — page 275. • Case data fields can be included in your sort criteria, although it causes you to take a performance hit because the TIBCO iProcess Objects Server must get case data from the engine.

See "Sorting Work Items and Cases" on page 273 for specific information about setting up sort criteria.

Getting Audit Data

If audit data is requested on the cases in the pageable list, this causes the TIBCO iProcess Objects Server to retrieve the audit data from the engine for each case on which it's requested. This impacts the performance of a case filter operation. (See "Getting Audit Step Objects" on page 72 for information about how audit data is requested.) For efficiency reasons, only include audit data in cases in which it is really needed.

Filter Criteria Format

The following shows the valid format for your filter criteria expressions. This is a BNF-like description. A vertical line "|" indicates alternatives, and [brackets] indicate optional parts.

<criteria>

<exp> | <exp> <logical_op> <exp> | [<criteria>]

<exp>

<value> <comparison_op> <value>

<logical_op>

and | or

<value>

<field> | <constant> | <systemfield>

<comparison_op>

= | <> | ? | < | > | <= | >=

<field>

<alpha>[fieldchars]

<systemfield>

See "System Fields Used in Filtering" on page 234 for a list of the allowable system fields.

<constant>

<date> | <time> | <numeric> | <string>

<date>

!<localdate>!

<time>

#<hour>:<min>#

<datetime>

"<localdate> <hour>:<min>"

<hour>

00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23

<min>

00| 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59

<localdate>

<mm>/<dd>/<yyyy> | <dd>/<mm>/<yyyy> | <yyyy>/<mm>/<dd> | <yyyy>/<dd>/<mm>

<mm>

01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12

<dd>

01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31

Note - The day and month portion of a date must be two digits. Correct: 09/05/2000. Incorrect: 9/5/2000.

<yyyy>

<digit> <digit> <digit> <digit>

<numeric>

<digits> [.<digits>]

<string>

"<asciichars>"

<asciichars>

<asciichar> [<asciichars>]

<asciichar>

ascii characters between values 32 and 126

<alpha>

a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<digit>

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<digits>

<digit> [<digits>]

<alphanum>

<alpha> | <digit>

<alphanums>

<alphanum> [<alphanums>]

<fieldchar>

<alpha> | <digit> | _

<fieldchars>

<fieldchar> [<fieldchars>]

System Fields Used in Filtering

System fields are symbolic references to data about a work item or case. These fields are primarily used by the TIBCO iProcess Engine (specifically, the Work Item Server) when performing filtering and sorting functions. The information that is available to the engine through the system fields is also available to the application though methods on work item and case objects. For example, **SW_CASENUM** is available to the client with the **vCaseId.getCaseNumber** method. The TIBCO iProcess Engine, however, doesn't have access to those methods, so the method names can't be used in filter and sort criteria — instead, the system field names need to be used in your expressions. For example:

```
"SW_CASENUM=5"
```

The system fields that are available for filtering are listed in the table below. Note that some system fields are only applicable for filtering on work items, some only for filtering on cases, and some are applicable to both (see the "Applies to" columns).

Filter Criteria	System Field	Data Type	Length	Applies to work items	Applies to cases
Addressee of work item (username@node)	SW_ADDRESSEE	Text	49	X	
Arrival date and time	SW_ARRIVAL	DateTime	16	X	
Arrival date	SW_ARRIVALDATE	Date	10	X	
Arrival time	SW_ARRIVALTIME	Time	5	X	
Case description	SW_CASEDESC	Text	24	X	X
Case ID in procedure	SW_CASEID	Numeric	7	X	
Case number	SW_CASENUM	Numeric	15	X	X
Case reference number	SW_CASEREF	Text	20	X	X
Date (current)	SW_DATE	Date	10	X	X
Deadline date and time	SW_DEADLINE	DateTime	16	X	
Deadline date	SW_DEADLINEDATE	Date	10	X	
Deadline expired flag (1 - expired; 0 - all work items)	SW_EXPIRED	Numeric	1	X	
Deadline set flag (1 - has deadline; 0 - all work items)	SW_HASDEADLINE	Numeric	1	X	
Deadline time	SW_DEADLINETIME	Time	5	X	
Forwardable work item flag (1 - forwardable; 0 - all work items)	SW_FWDABLE	Numeric	1	X	
Host name	SW_HOSTNAME	Text	24 or 8[a]	X	X
Locker of the work item (username)	SW_LOCKER	Text	24 or 8[a]	X	

Filter Criteria	System Field	Data Type	Length	Applies to work items	Applies to cases
Mail ID	SW_MAILID	String or Numeric [b]	7 (integer) 45 (string)	X	
Outstanding work item count (not available on TIBCO iProcess Engines)	SW_OUTSTANDCNT	Numeric	7		X
Pack file (not available on TIBCO iProcess Engines)	SW_PACKFILE	Text	13	X	
Priority of work item	SW_PRIORITY	Numeric	3	X	
Procedure description	SW_PRODESC	Text	24	X	X
Procedure name	SW_PRONAME	Text	8	X	X
Procedure number	SW_PRONUM	Numeric	7	X	X
Releasable work item (no input fields) (1 - releasable; 0 - all work items)	SW_RELABLE	Numeric	1	X	
Started date and time of the case	SW_STARTED	DateTime	16		X
Started date of the case	SW_STARTEDDATE	Date	10		X
Started time of the case	SW_STARTEDTIME	Time	5		X
Starter of the case (username@node)	SW_STARTER	Text	24 or 8[a]	X	X
Status of the case (“A” - active; “C” - closed)	SW_STATUS	Text	1		X
Step (work item) description	SW_STEPDESC	Text	24	X	
Step (work item) name	SW_STEPNAME	Text	8	X	
Step (work item) number in procedure	SW_STEPNUM	Numeric	7	X	
Suspended work item (1 - suspended; 0 - not suspended) (only available on TIBCO iProcess Engines)	SW_SUSPENDED	Numeric	1	X	
Terminated date and time of the case	SW_TERMINATED [c]	DateTime	16		X
Terminated date of the case	SW_TERMINATEDDATE [c]	Date	10		X
Terminated time of the case	SW_TERMINATEDTIME [c]	Time	5		X
Time (current)	SW_TIME	Time	5	X	X
Unopened work item (1 - unopened; 0 - all work items)	SW_NEW	Numeric	1	X	
Urgent flag (1- urgent; 0 - all work items)	SW_URGENT	Numeric	1	X	

Filter Criteria	System Field	Data Type	Length	Applies to work items	Applies to cases
Work queue parameter 1	SW_QPARAM1	Text	24	X	
Work queue parameter 2	SW_QPARAM2	Text	24	X	
Work queue parameter 3	SW_QPARAM3	Text	12	X	
Work queue parameter 4	SW_QPARAM4	Text	12	X	

a. This has a length of 24 for long-name systems, or 8 for short-name systems.

b. If using a TIBCO Process Engine, SW_MAILID is a numeric field of length 7; if using a TIBCO iProcess Engine, SW_MAILID is a string of length 45.

c. Only cases that have been terminated will be returned when filtering on these system fields. For instance, if your filter expression asks for cases where SW_TERMINATEDDATE < !09/01/2002!, only those cases that ARE terminated and whose termination date is earlier than 09/01/2002 are returned.

Note - The System Fields that can be set to 1 and 0 work in the following manner: When set to 1, only the respective work items are displayed; when set to 0, all work items are displayed. For example, if SW_FWDABLE is set to 1, this means "display only the forwardable work items". If it's set to 0, this means "don't display only the forwardable work items, instead, display all of them."

Data Types used in Filter Criteria

The following are definitions of the different data types used in filter criteria (see the Data Type column in the System Fields table in the previous section).

Data Type	Description
Numeric	Constant numbers are simply entered in the expression. Example: 425.00
Text	Text constants must be enclosed within double quotes. Example: "Smith"
Date	Date constants must be enclosed in exclamation marks. The ordering of the day, month and year is specified in the **staffpms** file (see "Date Format" on page 178). Example: !12/25/1997!
Time	Times can be included in the expression in the format hh:mm. They must be enclosed in pound signs. Uses the 24-hour clock. Example: #18:30#
DateTime	DateTime constants are a combination of a date and time, separated by a space, all enclosed in double quotes. The ordering of the day, month and year is specified in the **staffpms** file (see "Date Format" on page 178). Example: "12/25/1997 10:30"

Note - The day and month portion of a date must be two digits (correct: 09/05/2004; incorrect: 9/5/2004). The year portion of a date must be four digits (correct: 09/05/2004; incorrect: 09/05/04).

Data Type Conversions

If you specify a filter expression that compares values of data with different types, the following conversion takes place:

Filtering Work Items on the WIS

If comparing data of different types when filtering work items, the WIS will do the following:

- If comparing a string to any other data type (e.g., String = Numeric, String = Date, etc.), the WIS will attempt to convert the string to the non-string data type, then the comparison is performed. If the string cannot be converted to the non-string data type (for example, you are comparing a string to a Date, but the string value does not fit in the Date format), a syntax error is thrown.
- If comparing any other mismatched data types (e.g., Numeric = Date, Time = Date, etc.), the comparison will return a False.

Filtering Cases on the TIBCO iProcess Objects Server

If comparing data of different types when filtering cases, the TIBCO iProcess Objects Server will convert both data types to strings and compare their string values. See the examples below.

Example 1:

The expression:

```
!06/03/1999! < 34
```

will be converted to:

```
"06/03/1999" < "34"
```

Example 2:

Assume NUM_FIELD is a TIBCO field of type Numeric with a value of 275. The filter:

```
NUM_FIELD < "34"
```

will result in being true because NUM_FIELD will be converted to a string before the comparison is made ("275" < "34").

Filtering on Case Data Fields

You can filter work items in a work queue based on the values in the fields of the work item (referred to as "case data" fields).

There are two ways in which you can filter on case data:

- Using Case Data Queue Parameter (CDQP) Fields - CDQP fields are a more recent addition than Work Queue Parameter fields (see below) that allow you to filter and/or sort on an unlimited number of case data fields that appear in work items on your work queue.
- Using Work Queue Parameter Fields - These fields are used by assigning a case data field value to one of the pre-defined work queue parameter fields, then using the Work Queue Parameter field in filter or sort criteria. These fields have been superseded by CDQP fields as they were considered too limiting since there are only four of them.

More about CDQP and work queue parameter fields are described in the following subsections.

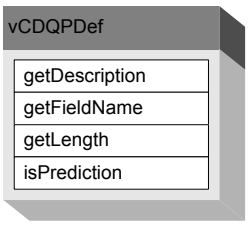
Note - With the WIS work item filtering enhancement, case data fields can be included in filter expressions only if they are defined as Case Data Queue Parameter (CDQP) fields. If your filter expression references a field that is not a CDQP for the queue, the WIS will return a syntax error, which causes the entire filter operation to fail. (The filter expression can also reference the Work Queue Parameter fields, which are essentially system fields — their names begin with "SW", e.g., SW_QPARAM1.)

Using Case Data Queue Parameter Fields

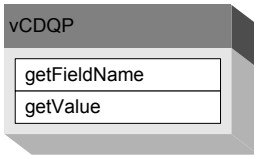
Case Data Queue Parameter (CDQP) fields provide an efficient method of filtering on the value of fields in your work items. To make use of this functionality, you must first pre-designate the fields you want to filter on as CDQP fields. Fields are designated as CDQP fields with the utility, **swutil**. This utility is used to create a list, on the TIBCO iProcess Engine, of the case data fields that are available to use for filtering. For information about using **swutil**, see the *TIBCO iProcess swutil and swbatch Reference Guide*.

Note - Case Data Queue Parameter fields are also used for efficiently sorting on case data, as described in the Sorting Work Items and Cases chapter.

Once you have created the list of CDQP fields with **swutil**, this list of fields is available with the **getCDQPDefs** method. This method, available on both the **sWorkQManager** and **vAWorkQ** objects, returns **vCDQPDef** objects, which provide the CDQP definitions. If called from **sWorkQManager**, it sends a message to the server to retrieve an array of **vCDQPDef** objects, one for each CDQP field defined on the specified work queue. If called from **vAWorkQ**, this returns an array of **vCDQPDef** objects, one for each CDQP field defined on the work queue represented by the local Value Object.



Once you have retrieved a work item Value Object (**vWorkItem**), the CDQP fields that are being used in that work item are available with the **getCDQPs** method. This method returns an array of **vCDQP** objects, one for each CDQP field that is being used in the work item. The **vCDQP** objects provide access to the values in the CDQP fields.



Your filter expressions can include any of the CDQP fields that have been defined on the work queue. For example, assuming LOAN_AMT is listed as one of the CDQP fields for the work queue, the following is a valid filter expression:

```
"LOAN_AMT = 500000"
```

CDQPs Contain Work Item Data

An important thing to understand is that when you filter (or sort) on the values in CDQPs, it's actually "work item data" in the CDQP (as opposed to "case data"). Work item data reflects any "keeps" that have been processed on the work item. In other words, if a user changes the value of a field, then keeps the work item, the CDQP for that field will reflect the changes the user made to the field. The "case data" is only updated when the work item is released.

See "Case Data vs. Work Item Data" on page 166 for more information.

Using Work Queue Parameter Fields

Note - Previous versions of the TIBCO iProcess Objects Server provided "Work Queue Parameter" fields that could be used for filtering and sorting work items based on the value of case data. Work Queue Parameter fields, however, did not provide the flexibility required by some customers. Therefore, a new method using "Case Data Queue Parameter" fields has been implemented (see the previous section). New development should use Case Data Queue Parameter fields to filter on case data instead of the Work Queue Parameter fields (Work Queue Parameter fields will continue to be supported, however).

"Work Queue Parameter" fields allow you to filter work items based on the value of case data fields in your client application. (Work Queue Parameter fields are also used for sorting on case data — see the *Sorting Work Items and Cases* chapter.)

If you have case/field data that you want to filter on (e.g., customer name, loan amount, etc.), it is much more efficient to assign the field value to one of the Work Queue Parameter fields, then filter on that field, instead of directly filtering on the application field. There are four work queue parameter fields available. The default definitions (which can be changed) for these fields are shown below:

Name	Type	Length	Description
SW_QPARAM1	Text	24	WQ Parameter Field 1
SW_QPARAM2	Text	24	WQ Parameter Field 2
SW_QPARAM3	Text	12	WQ Parameter Field 3
SW_QPARAM4	Text	12	WQ Parameter Field 4

These fields can be placed directly in forms, or you can assign the value of an application field to one of the work queue parameter fields through a script. For example:

```
SW_QPARAM1:=LAST_NAME
```

Then, you can filter on the value in the SW_QPARAM1 field. For example, to return only the work items that have a customer last name of Miller, the **FilterExpression** property is set as follows:

```
"SW_QPARAM1?\"Miller\""
```

This would be much more efficient then filtering on the LAST_NAME field.

The **vWorkItem** object contains methods that provide access to the values in the Work Queue Parameter fields — they are **getWorkQParam1 - getWorkQParam4**. These methods return the values you place in the system fields, SW_QPARAM1 - SW_QPARAM4, for each work item.

The **vWorkQ** object contains four methods that return a name for each of the Work Queue Parameter fields — they are **getWorkQParam1Name - getWorkQParam4Name**. If you use the TIBCO iProcess Workspace (Windows), these names appear in the column headers if you display the Work Queue Parameter fields in the Work Queue Manager. For information about modifying these names, see the *TIBCO iProcess Workspace (Windows) Manager's Guide*.

Work Queue Parameter Fields vs. Case Data Queue Parameter Fields

Why would you want to use the new Case Data Queue Parameter (CDQP) fields instead of the older Work Queue Parameter fields? The reasons for using each method is shown in the following table.

Case Data Filtering Method	Reasons For Using This Type
Work Queue Parameter Fields	• They are pre-configured, not requiring any administration (where as, CDQP fields require some additional administration). • They are available for all queues, requiring no additional administration. • They are already taking up resources (memory and disk space) whether they are used or not. (Adding four CDQP fields instead of using the already available Work Queue Parameter fields takes up additional resources.) • The load on the Work Item Server is slightly increased for each CDQP. • Configuring CDQP fields requires a TIBCO iProcess Engine shutdown.
Case Data Queue Parameter Fields	The primary reason to use CDQP fields is because if you use the four available Work Queue Parameter fields, then later realize you need more, it will require application changes — with CDQPs, you can just keep adding as many as needed.

Using Regular Expressions

Regular expressions may be included in filter expressions to provide powerful text search capabilities. They can be used when filtering either work items or cases. However, the way in which some regular expression special characters are evaluated differs between work items and cases. See the subsections below for information about the special characters that can be used with regular expressions when filtering work items and cases.

*Note - If using a regular expression when filtering predicted work items (**vPredictedItem** objects), the only special characters that can be used are the asterisk and question mark. They both work as wildcard characters, where the asterisk matches zero or more of any character, and the question mark matches any single character.*

All regular expressions must be in the following format:

constant ? "regular expression"

where:

- **constant** - A constant value or field name. If a field name is included in the expression, the field must be defined as a text data type (SWFieldType = swText). (Note that although the value in DateTime fields (e.g., SW_STARTED) is enclosed in quotes, they cannot be used with regular expressions, as they are not of text data type.)
- **?** - Special character signifying that a regular expression follows (interpreted as an equality operator).
- "**regular expression**" - Any valid regular expression (enclosed in double quotes).

Regular Expressions with Work Item Filtering

The following describes how regular expressions are evaluated when filtering work items (the Work Item Server (WIS) evaluates all work item filter expressions).

Note - If you are moving from a TIBCO iProcess Objects Server that does not have the WIS work item filtering enhancement (CR 12744) to one that does (see page 221 for more information), the way in which some regular expression special characters are evaluated will be different. This can result in a different set of work items being returned using the same filter expression.

A regular expression (RE) specifies a set of character strings. A member of this set of strings is "matched" by the RE. The REs allowed are:

The following one-character REs match a single character.

1. An ordinary character (not one of those discussed in number 2 below) is a one-character RE that matches itself *anywhere* in the constant/field. For example, an RE of “a” will match all constants/fields that contain “a”.
2. A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

   ., *, [, and \ — Period, asterisk, left square bracket, and backslash, respectively. These are always special, except when they appear within square brackets ([ ]; see Item 4 below).

   ^ — Caret or circumflex, which is special at the beginning of an entire RE, or when it immediately follows the left bracket of a pair of square brackets ([ ]) (see Item 4 below).

$ Dollar sign, which is special at the end of an entire RE. The character used to bound (i.e., delimit) an entire RE, which is special for that RE.

3. A period (**.**) is a one-character RE that matches any character except new-line.
4. A one-character RE followed by an asterisk (*) is an RE that matches zero or more occurrences of the one-character RE. If there is any choice, the longest, leftmost string that permits a match is chosen.
5. A non-empty string of characters enclosed in square brackets ([ ]) is a one-character RE that matches any one character in that string, with these additional rules:
   - If the first character of the string is a circumflex (^), the one-character RE matches any character except new-line and the remaining characters in the string. The ^ has this special meaning only if it occurs first in the string.
   - The minus (-) may be used to indicate a range of consecutive characters. For example, [0-9] is equivalent to [0123456789]. The minus sign loses this special meaning if it occurs first (after an initial ^, if any) or last in the string.
   - The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any). For example, [ ]a-f] matches either a right square bracket (]) or one of the ASCII letters a through f, inclusive.
   - The special characters ., *, [, and \ stand for themselves within such a string of characters.

The following rules may be used to construct REs from one-character REs:

1. A one-character RE is an RE that matches whatever the one-character RE matches.
2. The concatenation of REs is an RE that matches the concatenation of the strings matched by each component of the RE. For example, an RE of “abc” will match all constants/fields that contain “abc” anywhere in the constant/field.

An entire RE may be constrained to match only an initial segment or final segment of a line (or both):

1. A circumflex (^) at the beginning of an entire RE constrains that RE to match an initial segment of a line.
2. A dollar sign ($) at the end of an entire RE constrains that RE to match a final segment of a line.
3. The construction ^*entire RE*$ constrains the entire RE to match the entire line.

Regular Expressions with Case Filtering

The following describes how regular expressions are evaluated when filtering cases (the TIBCO iProcess Objects Server evaluates all case filter expressions).

A regular expression (RE) specifies a set of character strings. A member of this set of strings is "matched" by the RE.

The following one-character REs match a single character.

1. An ordinary character (not one of those discussed in number 2 below) is a one-character RE that matches itself. For example, an RE of “a” will match all constants/fields that match “a” exactly.

2. A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

   *, ?, [, and \ Asterisk, question mark, left square bracket, and backslash, respectively. These are always special, except when they appear within square brackets ([ ]; see Item 5 below).

3. An asterisk (*) is a one-character RE that matches zero or more of any character.
4. A question mark (?) is a one-character RE that matches any character except new-line.
5. A non-empty string of characters enclosed in square brackets ([ ]) is a one-character RE that matches any one character in that string, with these additional rules:
   - If the first character of the string is a circumflex (^), the one-character RE matches any character except new-line and the remaining characters in the string. The ^ has this special meaning only if it occurs first in the string.
   - The minus (-) may be used to indicate a range of consecutive characters. For example, [0-9] is equivalent to [0123456789]. The minus sign loses this special meaning if it occurs first (after an initial ^, if any) or last in the string.
   - The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any). For example, [ ]a-f] matches either a right square bracket (]) or one of the ASCII letters a through f, inclusive.
   - The special characters *, ?, [, and \ stand for themselves within such a string of characters.

The following rules may be used to construct REs from one-character REs:

1. A one-character RE is an RE that matches whatever the one-character RE matches.
2. The concatenation of REs is an RE that matches the concatenation of the strings matched by each component of the RE. For example, an RE of “abc” will match all constants/fields that contain “abc” exactly.

Using Escape Characters in the Filter Expression

Filter expressions require a string value. Therefore, if within the string value, you are required to provide another string, you must use an escape character to provide the quoted string within a string.

Use the back slash to indicate that the next character is a special character. In the example below, the back slashes indicate that the quotes that follow them are quoting the string "LOAN", and are not the ending quotes for the filter expression string.

```
"SW_PRONAME=\"LOAN\""
```

Filtering on Empty Fields

To filter on an empty field, you can use either of the following:

- compare the field with SW_NA, which checks to see if the field is "not assigned." For example:

```
"SOC_SEC_NUM=SW_NA"
```

- compare the field to an empty set of quotes. For example:

```
"SOC_SEC_NUM=\"\""
```

Note - See the previous section for information about using escape characters.

How to Specify Ranges of Values

Ranges of values can be specified in your filter expressions. This functionality, however, is limited to filtering on work items only — you cannot use range filtering when filtering cases.

Ranges must use the following format:

```
FilterField=[val1-val2|val3|val4-val5|.....|valn]
```

You can specify multiple ranges or single values, each separated by a vertical bar. The entire range expression is enclosed in square brackets. Only the '=' equality operator is allowed in a range filter expression.

Dates are specified as:

```
!dd/mm/yyyy!
```

*Note - The ordering of the day, month and year is specified in the **staffpms** file (see "Date Format" on page 178).*

Times are specified as:

```
#mm:hh#
```

DateTimes are specified as:

```
"dd/mm/yyyy mm:hh"
```

Range Filter Example 1:

This example returns the work items with case numbers between 50 and 100, and between 125 and 150, as well as the work item with case number 110:

```
SW_CASENUM=[50-100|110|125-150]
```

Range Filter Example 2:

To return all work items that arrived in the queue between 09/01/2000 and 09/03/2000 (inclusive), and that have a priority equal to 50:

```
SW_ARRIVALDATE=[!09/01/2000! - !09/03/2000!] AND SW_PRIORITY=50
```

Closing/Purging Cases Based on Filter Criteria

The **sCaseManager** object contains methods that allow you to close or purge cases based on filter criteria. These methods are:

- **closeCasesByCriteria** - This method closes cases that match the specified filter criteria. You must have case administration privilege to close a case (defined in TIBCO Business Studio). You also cannot close a case from a slave node.
- **purgeCasesByCriteria** - This method purges cases that match the specified filter criteria. You must have case administration privilege to purge a case (defined in TIBCO Business Studio). You also cannot purge a case from a slave node.

Both of these methods require a parameter that specifies a filter string expression. Use the filter expression syntax described in this chapter.

Closing and purging cases require that the user have system administrator authority (MENUNAME = ADMIN). See "User Attributes" on page 187 for information about the MENUNAME attribute.

Default Filter Criteria

The TIBCO iProcess Server Objects provide the ability to set *default* work item filter and sort criteria for a work queue.

Default criteria is a feature of the TIBCO iProcess Objects Server that allows you to save a specific criteria that persists for the work queue. Note, however, that the default criteria is not automatically applied to a pageable list of work items that is created for that work queue. Filter criteria must *always* be passed in the form of a **vWICriteria** Value Object when the list is requested. To apply the default criteria, you must call the **getDefaultCriteria** method (which returns a **vWICriteria** object representing the default criteria), then pass the **vWICriteria** object to the **getWorkItemList** or **getAWorkItemList** method when requesting the pageable list.

Note - If you use the TIBCO iProcess Workspace, filter criteria that are defined on the Work Queue Manager Work Item List Filter dialog become the default filter criteria for that work queue. The default filter criteria defined on this dialog can be viewed and/or affected by the methods described below.

The **sWorkQ** object contains methods that allow you to affect the default filter criteria (note that at the same time these methods are affecting the default sort criteria for the work queue — see "Setting Default Sort Criteria" on page 279):

- **changeDefaultCriteria** - This method sets the default criteria for the work queue based on the **vWICriteria** object passed in the method call. These filter criteria will persist on this work queue until changed again with this method or cleared with the **clearDefaultCriteria** method. (Note that this method is also setting the default sort criteria based on sort fields that are included in the **vWICriteria** object passed in the method call.)
- **clearDefaultCriteria** - This method clears the default filter criteria that were set either through the Work Queue Manager or by using the **changeDefaultCriteria** method. (Note that this also clears any default sort criteria that have been defined.)
- **getDefaultCriteria** - This method returns a **vWICriteria** object, indicating the currently set default criteria for the work queue. To apply default criteria, you must call this method to obtain the **vWICriteria** object, then pass that Value Object with the **getWorkItemList** or **getAWorkItemList** method when requesting the pageable list of work items.

You can **only** persist filter criteria that are a subset of those supported by the Work Queue Manager or an exception will be thrown when you execute **changeDefaultCriteria**. The following are the filter criteria that are supported by the Work Queue Manager.

System Field	Description
SW_ARRIVAL	Arrival date and time
SW_ARRIVALTIME	Arrival time
SW_ARRIVALDATE	Arrival date
SW_CASEDESC	Case description
SW_CASENUM	Case number
SW_CASEREF	Case reference number
SW_DEADLINE	Deadline date and time
SW_DEADLINETIME	Deadline time
SW_DEADLINEDATE	Deadline date
SW_EXPIRED	Deadline Expired Flag
SW_FWDABLE	Forwardable Items
SW_HASDEADLINE	Deadline Set Flag
SW_HOSTNAME	Host Name
SW_NEW	Unopened Work Item Flag
SW_PRIORITY	Priority of work item
SW_PRODESC	Procedure Description
SW_PRONAME	Procedure Name
SW_QPARAM1	Work Queue Parameter1
SW_QPARAM2	Work Queue Parameter2
SW_QPARAM3	Work Queue Parameter3
SW_QPARAM4	Work Queue Parameter4
SW_RELABLE	Releasable Work Item Flag
SW_STEPDESC	Form (Step) Description
SW_STEPNAME	Form (Step) Name
SW_URGENT	Urgent Work Item Flag

Also note the when using the **changeDefaultCriteria** method, your filter expressions must conform to the following guidelines:

- The *only* equality operator you can use is ‘=’.
- You cannot use any of the following equality operators: ‘?’,’<‘, ‘<=’, ‘>’, ‘>=’, and ‘<>’.
- You cannot use the OR logical operator.
- And since ‘?’ is not allowed, no regular expression syntax can be used.

15

Filtering Work Items and Cases

With WIS Work Item and Database Case Filtering

Important - Read this page first to determine which of the *Filtering Work Items and Cases* chapters you should use.

Over time, enhancements have been made to the TIBCO iProcess Objects Server to improve the efficiency of filtering and sorting work items and cases. Because the scope of the enhancements is fairly major, three chapters are now provided in this guide that describe how filtering and sorting work, depending on which of the enhancements have been implemented in your TIBCO iProcess Objects Server. Use the table below to determine which chapter to use, based on the enhancements in your TIBCO iProcess Objects Server.

Note - Although the topic of sorting is covered in a separate chapter, filtering and sorting is described as a single process in the Filtering Work Items and Cases chapters because that is the way it is performed — work items or cases are filtered, then the result set from the filter operation is sorted.

Two major enhancements have been added to the TIBCO iProcess Objects Server that impact filtering and sorting:

- **WIS Work Item Filtering** - This enhancement moved all work item filter processing to the Work Item Server (WIS). With this enhancement, all of the additional capabilities previously provided by the TIBCO iProcess Objects Server can now be performed by the WIS when filtering work items (such as allowing the OR logical operator, allowing the <, >, <=, >=, and <> operators, etc.). Since the WIS has the work items cached, and has direct access to case data, this provides for very efficient filtering and sorting of work items.

  Your server/engine must have the following CRs implemented for this enhancement: TIBCO iProcess Objects Server - CR 12744; TIBCO Process/iProcess Engine - CR 12686.

- **Database Case Filtering** - This enhancement moved all case filter and sort processing to the database. With this enhancement, the filter expression is translated into an SQL select statement, which is used to create the result set from the cases in the database. The result set is then sorted. Because of the indexing ability of the database, this provides for very efficient filtering and sorting of cases.

  This enhancement was implemented in the following CRs: TIBCO iProcess Objects Server - CR 13182; TIBCO Process/iProcess Engine - CR 13098.

Use the following table to determine which of the *Filtering Work Items and Cases* chapters to use:

If your TIBCO iProcess Objects Server includes...	Use this chapter...
Neither of the enhancements listed above	Chapter 13
Only the WIS Work Item Filtering enhancement (CR 12744)	Chapter 14
Both the WIS Work Item Filtering and the Database Case Filtering enhancements (CRs 12744 and 13182)	Chapter 15

Introduction

The TIBCO iProcess Server Objects provide the ability to *filter* work items and cases, allowing you to filter out all those you aren't currently interested in. For example, you may only be interested in the work items that arrived in the work queue today, in which case you could specify a filter expression that filters out all work items other than those that arrived today:

```
"SW_ARRIVALDATE = !08/02/2001!"
```

The benefits of this are two-fold:

- It allows you to display to the user only those cases or work items that are of interest to them.
- It reduces the amount of work the client and the server need to do. When the result set from the filter operation results in fewer work items or cases, this reduces the work load on the client and server.

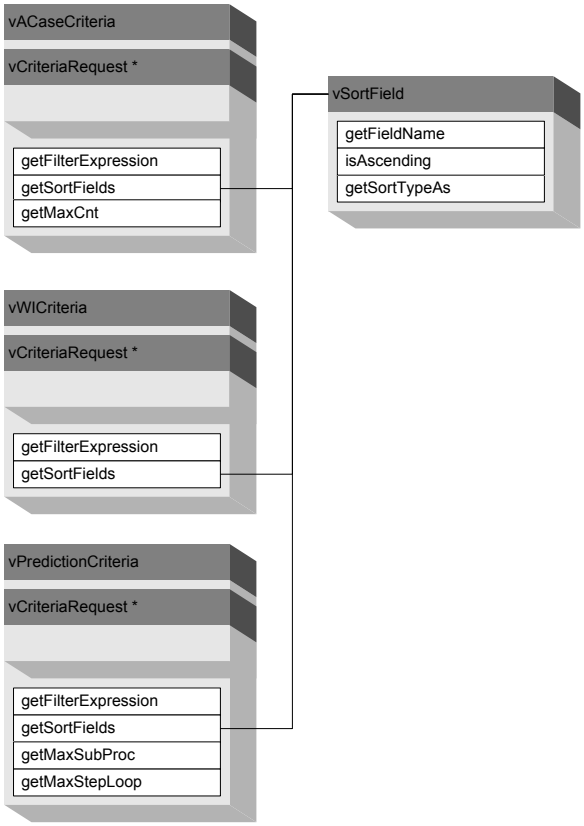
Specifying Filter Criteria

The TIBCO iProcess Server Objects provide three "criteria" objects to specify filter criteria:

- **vACaseCriteria** (for cases)
- **vWICriteria** (for work items)
- **vPredictionCriteria** (for predicted work items) (Note that since predicted items are stored in the database, they are filtered in the same way as cases when you have the database case filtering enhancement — see "Filtering/Sorting Cases" on page 255 for more information.)

Note that these criteria objects are used for both filtering and sorting work items and cases — see "Sorting Work Items and Cases" on page 273 for information about sorting.

Work items and cases that can be filtered are always returned in pageable lists (**sPageableList**, **sPageableListR**, or **sPageableListJ** objects) — see "Working with Lists" on page 137 for information about using pageable lists after the filtered work items or cases have been returned from the server.



The methods that return filtered work items and cases are summarized in the table below.

Method	Uses this Criteria Object	Returns Pageable List of this Object
sWorkQ.getWorkItemList	vWICriteria	vWorkItem
sWorkQ.getAWorkItemList	vWICriteria	vAWorkItem
sCaseManager.getACaseList	vACaseCriteria	vACase
sCaseManager.getPredictedItemList	vPredictionCriteria	vPredictedItem

To specify filter criteria for a pageable list of work items or cases, follow these steps:

1. Construct a **vACaseCriteria**, **vWICriteria**, or **vPredictionCriteria** object, setting the *aFilterExpression* parameter to the desired filter expression string — see the "Defining Filter Expressions" section below for specifics about creating these strings.
2. Pass the **vACaseCriteria**, **vWICriteria**, **vPredictionCriteria** object as an input parameter with one of the methods listed in the table above, depending on the type of object you want returned in the pageable list.

Defining Filter Expressions

To filter work items, cases, or predicted work items, you must define a *filter expression* string, which is passed in the *aFilterExpression* parameter with the applicable criteria object (see the previous subsection). The filter expression string is evaluated against each work item or case you are requesting, returning either True or False. If it returns True, the work item/case is included in the pageable list; if it returns False, the work item/case is not included in the pageable list. For example, you may choose to filter the list to include only work items with the urgent flag set ("SW_URGENT = True").

Filter expression strings can contain elements such as system fields (SW_CASENUM, SW_NEW, etc.), logical operators (AND, OR), comparison operators (=, <, <=, etc.), ranges of values, etc. Details about filter expressions is described in the subsections that follow.

Note that the left and right side of comparison operators (=, <, >, <=, >=, <>, ?) must each consist of only a single field name or single constant. It cannot be an expression containing operators (+, -, /, *, etc.).

Example 1:
To define a filter for work items matching all new items from procedure LOANS, set the filter expression to the following string:

```
"SW_NEW=1 AND SW_PRONAME=\"LOANS\""
```

Example 2:
To define a filter for all work items that arrived after June 20, 2001 (assuming mm/dd/yyyy date locale setting in the engine), set the filter expression to the following string:

```
"SW_ARRIVALDATE > !06/20/2001!"
```

Example 3:

To define a filter for all cases with field LOAN_AMT having a value greater than 100000, set the filter expression to the following string:

```
"LOAN_AMT > 100000"
```

Length of Filter Expressions

The exact length that you can make filter expressions is not well defined, although some approximations are provided below. The filter expression length depends on whether you are filtering work items or cases, as follows:

- **Work Items -** The filter expression created is converted into a SAL-compatible expression. The maximum size after conversion is 2K bytes. Note, however, that the conversion can increase the size of the expression. Tests have shown that a 1700-byte expression increases to approximately 2K bytes during the conversion. The amount of increase depends on the operators used in the expression.
- **Cases -** Filter expressions for cases are also converted into a SAL-compatible expression as described above for work items. These filter expressions then undergo another conversion to SQL Select statements. This second conversion can dramatically increase the size of the expression by anywhere between two to four times. The maximum size of the expression after the conversion to the SQL Select statement is 4K bytes.

  If an expression filtering cases exceeds the maximum size after the conversion to a SQL Select statement, an “Error in expression syntax” is returned to the client (note that the message is not descriptive of the problem).

Large Filter Expressions May Require Larger Stack Size

When filtering cases, if the filter expression contains a large number of clauses, the SAL **sal_xpc_list_filter_cases** routine crashes, resulting in the TIBCO iProcess Objects Server crashing (a clause consists of a field being compared to a value, e.g., “SW_CASENUM = 7 AND AMOUNT > 10000” contains two clauses).

If you experience this type of error condition, increasing the thread stack size may resolve the problem. This is done using the **StackSize** configuration parameter. The default stack size is 700KB per thread in AIX, HP-UX, and Linux, and 1M in Solaris. To increase this value, you must manually add the **StackSize** parameter to your configuration file ($SWDIR/**seo/data/swentobjsv.cfg**). This parameter allows you to specify the number of kilobytes to set the stack size, per thread. Note that increasing the stack size will increase the overall memory size of the TIBCO iProcess Objects Server. For more information, see **StackSize** in the *TIBCO iProcess Objects Server Administrator’s Guide.*

Number of Cases or Work Items in a Filtered Pageable List

There are a number of methods available that provide information about the number of cases or work items on a filtered pageable list. They are:

On **sPageableList**:

- **Available Count (getAvailableCnt) -** This returns the total number of items available in the indexed collection on the TIBCO iProcess Objects Server. This count includes only the work items or cases that satisfy the filter criteria.

- **Local Count (getLocalCnt)** - This returns the number of cases or work items currently being held in the local bock(s) on the sPageableList. If **isKeepLocalItems** has been set to False, this count will always be less than or equal to the number of items per block (**getItemsPerBlock**).

Note - The counts above require an understanding of how an indexed collection is created on the TIBCO iProcess Objects Server, and how blocks of objects are held locally in the pageable list. See *"Working with Lists" on page 137* *for information.*

On **vSummary**:

- **Exclude Count (getExcludeCnt)** - When filtering work items, this method returns the number of work items that did not satisfy the Boolean expression specified in the filter expression, and therefore, were not included in the pageable list.

  When filtering cases, this method is no longer applicable (it returns -1 if the pageable contains cases). Since the filter processing is being handled in the database, determining an invalid count would require the database to determine the row count, which would decrease the performance improvement gained by the database creating the result set.
- **Invalid Count (getInvalidCnt)** - This method is no longer applicable when filtering work items or cases (it always returns -1).
- **Over Maximum Count (getOverMaxCnt)** - This returns the number of cases that were *not* returned from the server because the number returned was limited using the *aMaxCnt* parameter on the **vACaseCriteria** constructor when retrieving a list of cases. This is applicable only when retreiving cases.

On **vWISummary**:

These counts are applicable only if the pageable list contains work items.

- **Urgent Count (getUrgentCnt)** - This returns the number of work items on the pageable list that are marked as urgent. A work item is marked as urgent if its priority value (vWorkItem.getPriorty) is less than or equal to a specific value. By default, the value is 10, although it can be modified in the **staffcfg** file.
- **Deadline Count (getDeadlineCnt)** - This returns the number of work items on the pageable list that have deadlines.
- **Unopened Count (getUnopenedCnt)** - This returns the number of work items on the pageable list that have not been opened (locked).

Filtering/Sorting in an Efficient Manner

The way in which you write your filter expressions can have an affect on how efficiently they are evaluated. This section provides guidelines about what types of elements you can include in your filter expressions (and those you should avoid) to ensure an efficient filter operation.

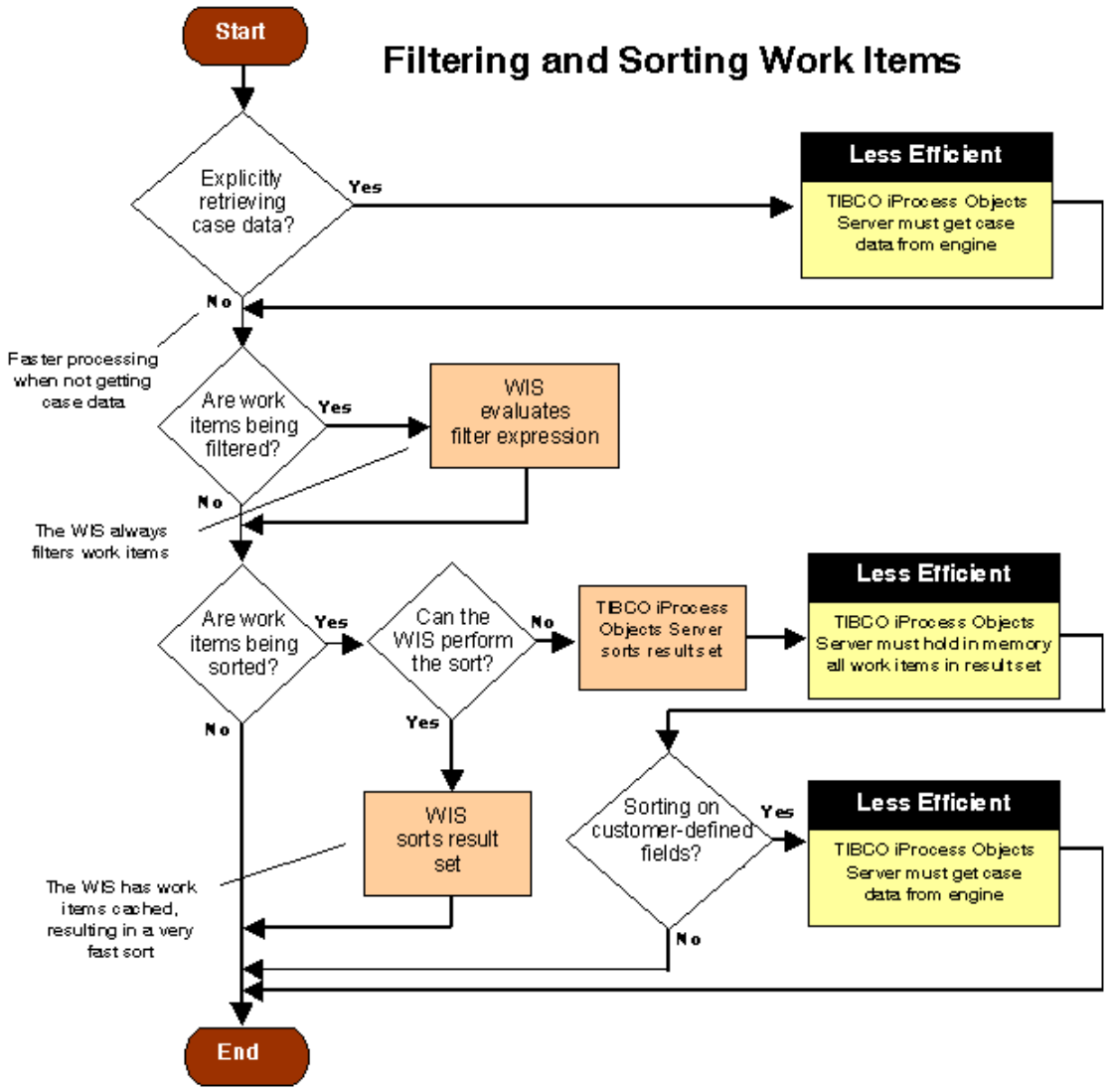
Flow diagrams (one for work items; one for cases) are shown in the following subsections that illustrate the decision process that takes place during a filter/sort operation. Note that the flow diagrams show filtering and sorting taking place in a single operation; that is the way filtering and sorting is processed — works items or cases are filtered to create a result set, then the result set is sorted. The flow diagrams also illustrate how to prevent the filter/sort operation from being less efficient.

Filtering/Sorting Work Items

When filtering and sorting work items:

- Work items are *always* filtered by the Work Item Server (WIS). The WIS has work items cached in memory, allowing it to evaluate filter expressions for work items very quickly. The elements you are allowed to use in your filter expressions are listed in "Work Items are Filtered by the WIS" on page 253.
- If you "get case data" in your application, this causes the filter processing to be less efficient. More about "getting case data" is explained below.
- Work items can be sorted by either the WIS or the TIBCO iProcess Objects Server, depending on how you specify the sort criteria. It's preferable to have the WIS sort the result set from the filter operation. This is explained in detail below.

The following flow diagram shows the decision process that takes place when filtering and sorting work items.



As shown in the illustration, there are a couple of actions that will cause the filter/sort operation to be less efficient when filtering and sorting work items:

- Getting case data
- Performing the sort operation on the TIBCO iProcess Objects Server

Additional information about these actions is provided in the subsections that follow.

Getting Case Data

Causing the server to "get case data" means that the TIBCO iProcess Objects Server must retrieve case data from the TIBCO iProcess Engine. This significantly slows down the filter/sort processing. The following actions cause the TIBCO iProcess Objects Server to get case data:

- Explicitly Retrieving Case Data - If you explicitly ask for case data using the *aCaseFieldNames* parameter when constructing a work item "content" object (**vWIContent**), the server will explicitly retrieve the data in those fields from the engine. See "Retrieving Field Data from the Server" on page 167 for information about the use of the *aCaseFieldNames* parameter.
- Having the TIBCO iProcess Objects Server sort on customer-defined fields - The TIBCO iProcess Objects Server does not have direct access to case data. Therefore, if the sort operation is being handled by the TIBCO iProcess Objects Server, and you sort on a customer-defined field (i.e., any field on a form that is not a system field (SW_PRIORITY, SW_PRONAME, etc.)), the TIBCO iProcess Objects Server must retrieve the data in that field from the engine, adversely affecting performance.

Note that although the flow diagram shows that there are two different places where you can take a performance hit by getting case data, the actual hit only occurs once, i.e., you don't take two performance hits, for instance, if you explicitly retrieve case data *and* the TIBCO iProcess Objects Server is sorting on customer-defined fields; the TIBCO iProcess Objects Server only has to get case data once for the entire operation.

Work Items are Filtered by the WIS

As shown in the *Filtering and Sorting Work Items* illustration, work items are always filtered by the WIS. The WIS has work items cached in memory, allowing it to evaluate filter expressions for work items very quickly.

The following table lists the elements that can be used in filter expressions when filtering work items:

Element	Description
Comparison Operators	=, <, >, <=, >=, <> (The ? character can also be used as an equality operator with regular expressions — see "Using Regular Expressions" on page 267.)
Logical Operators	AND, OR
System Fields	All system fields that are applicable to work items (see the *Applies To* column in the table of system fields used for filtering — page 260)
Parentheses	Parentheses can be used to construct more complex filter expressions, or to make your expressions more readable.

Element	Description
Case Data Fields	Case data fields can be included in your filter expressions ONLY if they are first defined as CDQPs. If your filter expression references a field that is not a CDQP, the WIS will return a syntax error, which causes the entire filter operation to fail. See "Filtering on Case Data Fields" on page 264 for information.
Wildcards	The wildcard characters '*' and '?' as part of a string on equality checks. The '*' character matches zero or more of any character. The '?' character matches any single character.
Ranges of Values	Ranges of values can be included in your work item filter expressions by using a specific syntax — see "How to Specify Ranges of Values" on page 269 for information.
Regular Expressions	Regular expressions can be used when filtering work items, allowing you to do complex pattern matching. See "Using Regular Expressions" on page 267.

The following are examples of filter expressions for filtering work items:

- To define a filter for all unopened work items, set the filter expression to:

  ```
  "SW_NEW = 1"
  ```

- To define a filter for work items that either arrived in the work queue after 03/01/2005, or that arrived on or before 02/20/2005 *and* have not been opened yet:

  ```
  "SW_ARRIVALDATE > !03/01/2005! OR (SW_ARRIVALDATE <= !02/20/2005! AND
  SW_NEW = 1)"
  ```

Can the WIS Perform the Sort Operation?

Work items can be sorted by either the WIS or the TIBCO iProcess Objects Server, depending on the sort criteria you use. Whenever possible, you should use the sort criteria that can be evaluated by the WIS. If the TIBCO iProcess Objects Server must perform the sort operation, **it must hold in memory all work items in the filter result set**. If the result set from the filter operation is very large, this can consume a significant amount of memory.

The table below shows the sort criteria you can use to cause the sort operation to be performed by the WIS. It also lists the expanded criteria available by the TIBCO iProcess Objects Server. Using this expanded criteria causes the sort operation to be performed by the TIBCO iProcess Objects Server, which is less efficient because it must hold the result set in memory.

Sort Criteria the WIS can Process
• System fields that are "WIS-compatible". See the WIS-compatible column in the table of *System Fields used in Sorting* on page 275. (The system fields must be applicable to filtering work items.) • Case Data Queue Parameter (CDQP) fields. See "Sorting on Case Data Fields" on page 277 for more information.
Sort Criteria the TIBCO iProcess Objects Server must Process
• System fields that are NOT "WIS-compatible". See the WIS-compatible column in the table of *System Fields used in Sorting* on page 275. (The system fields must be applicable to filtering work items.) • Case data fields that have NOT been designated as Case Data Queue Parameter (CDQP) fields. See "Sorting on Case Data Fields" on page 277 for more information.

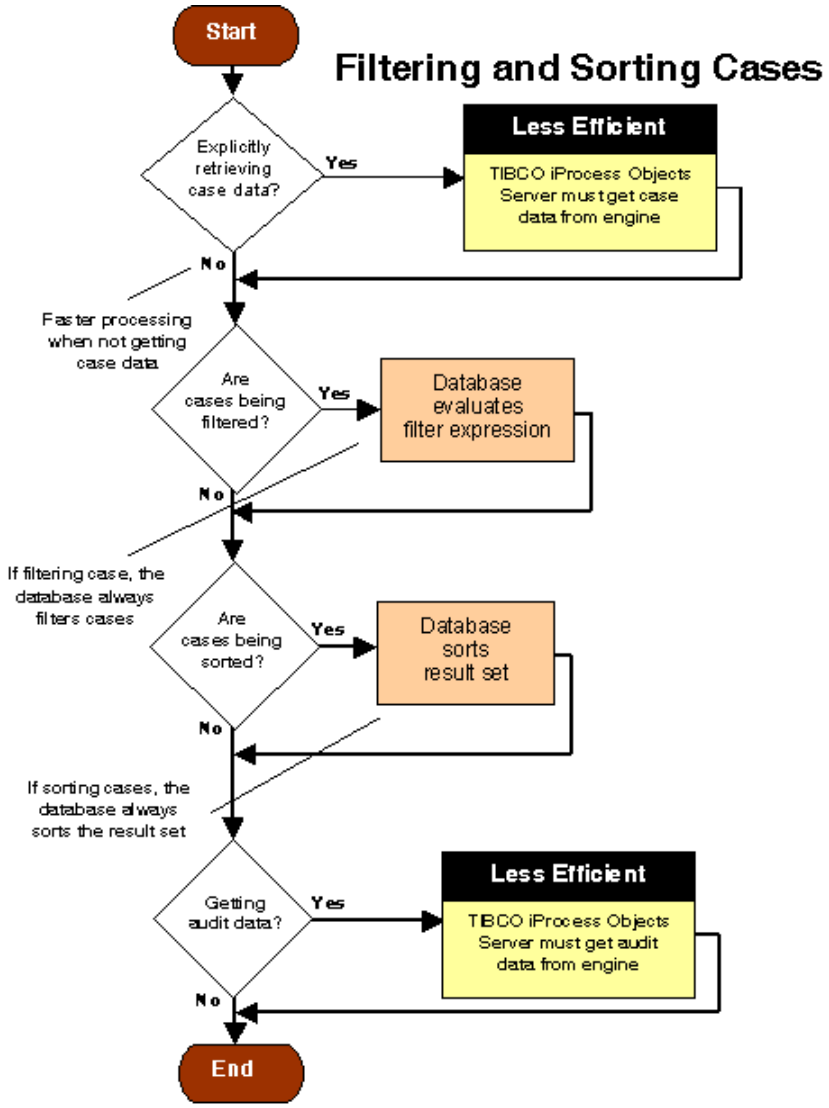
See "Sorting Work Items and Cases" on page 273 for information about setting up sort criteria.

Filtering/Sorting Cases

When filtering and sorting cases:

- Cases are *always* filtered by the database. The filter expression is translated into an SQL select statement, which is used to create the result set from the cases in the database. Because of the indexing ability of the database, this provides for very efficient filtering of cases. The elements you are allowed to use in your filter expressions to filter cases are listed in "The Database Filters Cases" on page 257.
- If you "get case data" in your application, this causes the filter processing to be less efficient. More about "getting case data" is explained below.
- Cases are *always* sorted by the database. The result set from the filter operation (if performed) is sorted in the database. See the table in "The Database Sorts Cases" on page 258 for the sort criteria that can be used when sorting cases.

The following flow diagram shows the decision process that takes place when filtering and sorting cases.



As shown in the illustration, there are some actions you should avoid, if possible, when filtering and sorting cases:

- Getting case data
- Getting audit data

Additional information about these actions is provided in the subsections that follow.

Getting Case Data

Causing the server to "get case data" means that the TIBCO iProcess Objects Server must retrieve case data from the TIBCO iProcess Engine. This significantly slows down the filter/sort processing. The following action causes the TIBCO iProcess Objects Server to get case data:

- Explicitly Retrieving Case Data - If you explicitly ask for case data using the *aCaseFieldNames* parameter when constructing a case "content" object (**vACaseContent**), the server will explicitly retrieve the data in those fields from the engine. See "Retrieving Field Data from the Server" on page 167 for information about the use of the *aCaseFieldNames* parameter.

The Database Filters Cases

Cases are always filtered by the database. The filter expression is translated into an SQL select statement, which is used to create the result set from the cases in the database. Because of the indexing ability of the database, this provides for very efficient filtering of cases.

The following table lists the elements that can be used in filter expressions when filtering cases:

Element	Description
Logical Operators	AND, OR
Comparison Operators	=, <, >, <=, >=, <> (The ? character can also be used as an equality operator with regular expressions — see "Using Regular Expressions" on page 267.)
System Fields	All system fields that are applicable to cases (see the *Applies To* column in the table of system fields used for filtering — page 260).
Parentheses	Parentheses can be used to construct more complex filter expressions, or to make your expressions more readable.
Case Data Fields	Case data fields can be included in your filter expressions (although field-to-field comparisons (e.g., FIELD1 = FIELD2, FIELD1> FIELD2, etc.) are not supported).
Wildcards	The wildcard characters '*' and '?' as part of a string on equality checks. The '*' character matches zero or more of any character. The '?' character matches any single character.
Regular Expression	Regular expressions can be used when filtering cases. However, when using the regular expression equality operator (?) in your filter expression, the regular expression string can include the * and ? wildcard characters, but none of the other regular expression special characters (the database is not able to interpret the other special characters). See "Using Regular Expressions" on page 267.

The following are examples of filter expressions for filtering cases:

- To define a filter for all cases that were started on or before March 1, 2003 (assume mm/dd/yyyy date locale setting in the engine), set the filter expression to:

```
"SW_STARTEDDATE <= !03/01/2003!"
```

- To define a filter for cases with a case number of 1, or a case number of 2 *and* a case description of "Test":

```
"SW_CASENUM = 1 OR (SW_CASENUM = 2 AND SW_CASEDESC = \"Test\")"
```

The Database Sorts Cases

When sorting cases in the database, the following sort criteria can be used:

Sort Criteria for Sorting Cases
• All system fields that are applicable to cases (see the *Applies To* column in the table of system fields used for sorting — page 273). • Case data fields can be included in your sort criteria when sorting cases.

See "Sorting Work Items and Cases" on page 273 for specific information about setting up sort criteria.

Getting Audit Data

If audit data is requested on the cases in the pageable list, this causes the TIBCO iProcess Objects Server to retrieve the audit data from the engine for each case on which it's requested. This impacts the performance of a case filter operation. (See "Getting Audit Step Objects" on page 72 for information about how audit data is requested.) For efficiency reasons, only include audit data in cases in which it is really needed.

Filter Criteria Format

The following shows the valid format for your filter criteria expressions. This is a BNF-like description. A vertical line "|" indicates alternatives, and [brackets] indicate optional parts.

<criteria>

<exp> | <exp> <logical_op> <exp> | [<criteria>]

<exp>

<value> <comparison_op> <value>

<logical_op>

and | or

<value>

<field> | <constant> | <systemfield>

<comparison_op>

= | <> | ? | < | > | <= | >=

<field>

<alpha>[fieldchars]

<systemfield>

See "System Fields Used in Filtering" on page 260 for a list of the allowable system fields.

<constant>

<date> | <time> | <numeric> | <string>

<date>

!<localdate>!

<time>

#<hour>:<min>#

<datetime>

"<localdate> <hour>:<min>"

<hour>

00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23

<min>

00| 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59

<localdate>

<mm>/<dd>/<yyyy> | <dd>/<mm>/<yyyy> | <yyyy>/<mm>/<dd> | <yyyy>/<dd>/<mm>

<mm>

01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12

<dd>

01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31

Note - The day and month portion of a date must be two digits. Correct: 09/05/2000. Incorrect: 9/5/2000.

<yyyy>

<digit> <digit> <digit> <digit>

<numeric>

<digits> [.<digits>]

<string>

"<asciichars>"

<asciichars>

<asciichar> [<asciichars>]

<asciichar>

ascii characters between values 32 and 126

<alpha>

a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<digit>

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<digits>

<digit> [<digits>]

<alphanum>

<alpha> | <digit>

<alphanums>

<alphanum> [<alphanums>]

<fieldchar>

<alpha> | <digit> | _

<fieldchars>

<fieldchar> [<fieldchars>]

System Fields Used in Filtering

System fields are symbolic references to data about a work item or case. These fields are primarily used by the TIBCO iProcess Engine (specifically, the Work Item Server) when performing filtering and sorting functions. The information that is available to the engine through the system fields is also available to the application though methods on work item and case objects. For example, **SW_CASENUM** is available to the client with the **vCaseId.getCaseNumber** method. The TIBCO iProcess Engine, however, doesn't have access to those methods, so the method names can't be used in filter and sort criteria — instead, the system field names need to be used in your expressions. For example:

```
"SW_CASENUM=5"
```

The system fields that are available for filtering are listed in the table below. Note that some system fields are only applicable for filtering on work items, some only for filtering on cases, and some are applicable to both (see the "Applies to" columns).

Filter Criteria	System Field	Data Type	Length	Applies to work items	Applies to cases
Addressee of work item (username@node)	SW_ADDRESSEE	Text	49	X	
Arrival date and time	SW_ARRIVAL	DateTime	16	X	
Arrival date	SW_ARRIVALDATE	Date	10	X	
Arrival time	SW_ARRIVALTIME	Time	5	X	
Case description	SW_CASEDESC	Text	24	X	X
Case ID in procedure	SW_CASEID	Numeric	7	X	
Case number	SW_CASENUM	Numeric	15	X	X
Case reference number	SW_CASEREF	Text	20	X	X
Date (current)	SW_DATE	Date	10	X	X
Deadline date and time	SW_DEADLINE	DateTime	16	X	
Deadline date	SW_DEADLINEDATE	Date	10	X	
Deadline expired flag (1 - expired deadline; 0 - no expired deadline)	SW_EXPIRED	Numeric	1	X	
Deadline set flag (1 - has deadline; 0 - no deadline)	SW_HASDEADLINE	Numeric	1	X	

Filter Criteria	System Field	Data Type	Length	Applies to work items	Applies to cases
Deadline time	SW_DEADLINETIME	Time	5	X	
Forwardable work item flag (1 - forwardable; 0 - not forwardable)	SW_FWDABLE	Numeric	1	X	
Host name	SW_HOSTNAME	Text	24 or 8[a]	X	
Locker of the work item (username)	SW_LOCKER	Text	24 or 8[a]	X	
Mail ID	SW_MAILID	String or Numeric [b]	7 (integer) 45 (string)	X	
Outstanding work item count (not available on TIBCO iPro-cess Engines)	SW_OUTSTANDCNT	Numeric	7		X
Pack file (not available on TIBCO iProcess Engines)	SW_PACKFILE	Text	13	X	
Priority of work item	SW_PRIORITY	Numeric	3	X	
Procedure description	SW_PRODESC	Text	24	X	X
Procedure name	SW_PRONAME	Text	8	X	X
Procedure number	SW_PRONUM	Numeric	7	X	X
Releasable work item (no input fields) (1 - releasable; 0 - not releaseable)	SW_RELABLE	Numeric	1	X	
Started date and time of the case	SW_STARTED	DateTime	16		X
Started date of the case	SW_STARTEDDATE	Date	10		X
Started time of the case	SW_STARTEDTIME	Time	5		X
Starter of the case (username@node)	SW_STARTER	Text	24 or 8[a]	X	X
Status of the case (“A” - active; “C” - closed)	SW_STATUS	Text	1		X
Step (work item) description	SW_STEPDESC	Text	24	X	
Step (work item) name	SW_STEPNAME	Text	8	X	
Step (work item) number in pro-cedure	SW_STEPNUM	Numeric	7	X	
Suspended work item (1 - suspended; 0 - not suspended) (only available on TIBCO iPro-cess Engines)	SW_SUSPENDED	Numeric	1	X	
Terminated date and time of the case	SW_TERMINATED [c]	DateTime	16		X
Terminated date of the case	SW_TERMINATEDDATE [c]	Date	10		X

Filter Criteria	System Field	Data Type	Length	Applies to work items	Applies to cases
Terminated time of the case	SW_TERMINATEDTIME [c]	Time	5		X
Time (current)	SW_TIME	Time	5	X	X
Unopened work item (1 - unopened; 0 - have been opened)	SW_NEW	Numeric	1	X	
Urgent flag (1- urgent; 0 - not flagged urgent)	SW_URGENT	Numeric	1	X	
Work queue parameter 1	SW_QPARAM1	Text	24	X	
Work queue parameter 2	SW_QPARAM2	Text	24	X	
Work queue parameter 3	SW_QPARAM3	Text	12	X	
Work queue parameter 4	SW_QPARAM4	Text	12	X	

a. This has a length of 24 for long-name systems, or 8 for short-name systems.

b. If using a TIBCO Process Engine, SW_MAILID is a numeric field of length 7; if using a TIBCO iProcess Engine, SW_MAILID is a string of length 45.

c. Only cases that have been terminated will be returned when filtering on these system fields. For instance, if your filter expression asks for cases where SW_TERMINATEDDATE < !09/01/2002!, only those cases that ARE terminated and whose termination date is earlier than 09/01/2002 are returned.

Data Types used in Filter Criteria

The following are definitions of the different data types used in filter criteria (see the Data Type column in the System Fields table in the previous section).

Data Type	Description
Numeric	Constant numbers are simply entered in the expression. Example: 425.00
Text	Text constants must be enclosed within double quotes. Example: "Smith"
Date	Date constants must be enclosed in exclamation marks. The ordering of the day, month and year is specified in the **staffpms** file (see "Date Format" on page 178). Example: !12/25/1997!
Time	Times can be included in the expression in the format hh:mm. They must be enclosed in pound signs. Uses the 24-hour clock. Example: #18:30#
DateTime	DateTime constants are a combination of a date and time, separated by a space, all enclosed in double quotes. The ordering of the day, month and year is specified in the **staffpms** file (see "Date Format" on page 178). Example: "12/25/1997 10:30"

Note - The day and month portion of a date must be two digits (correct: 09/05/2004; incorrect: 9/5/2004). The year portion of a date must be four digits (correct: 09/05/2004; incorrect: 09/05/04).

Data Type Conversions

If you specify a filter expression that compares values of data with different types, the following conversion takes place (this applies to both work items and cases that are filtered by either the WIS or the database):

- If comparing a string to any other data type (e.g., String = Numeric, String = Date, etc.), the WIS/database will attempt to convert the string to the non-string data type, then the comparison is performed. If the string cannot be converted to the non-string data type (for example, you are comparing a string to a Date, but the string value does not fit in the Date format), a syntax error is thrown.
- If comparing any other mismatched data types (e.g., Numeric = Date, Time = Date, etc.), the comparison will return a False.

Filtering on Case Data Fields

When filtering **cases**, you can include case data field names in your filter expression. Note that the values in these fields contain "case data" (as opposed to "work item data"). "Case data" is updated with the values that have been entered into the fields of a work item only when you "release" that work item. For more information about case data and work item data, see "Case Data vs. Work Item Data" on page 166.

When filtering **work items**, you can include case data fields in your filter expression only if they have been designated as one of the following:

- **Case Data Queue Parameter (CDQP) Fields** - CDQP fields are a more recent addition than Work Queue Parameter fields (see below) that allow you to filter and/or sort on an unlimited number of case data fields that appear in work items on your work queue.
- **Work Queue Parameter Fields** - These fields are used by assigning a case data field value to one of the pre-defined work queue parameter fields, then using the Work Queue Parameter field in filter or sort criteria. These fields have been superseded by CDQP fields as they were considered too limiting since there are only four of them.

If your filter expression references a field that is not a CDQP or a Work Queue Parameter field, the WIS will return a syntax error, which causes the entire filter operation to fail.

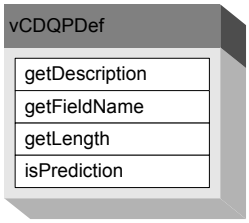
More about CDQP and work queue parameter fields are described in the following subsections.

Using Case Data Queue Parameter Fields

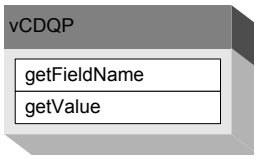
Case Data Queue Parameter (CDQP) fields provide an efficient method of filtering on the value of fields in your work items. To make use of this functionality, you must first pre-designate the fields you want to filter on as CDQP fields, then map them to a specific work queue. For information about this, see the *TIBCO iProcess swutil and swbatch Reference Guide*.

Note - Case Data Queue Parameter fields are also used for efficiently sorting on case data, as described in the Sorting Work Items and Cases chapter.

Once you have created the list of CDQP fields with **swutil**, this list of fields is available with the **getCDQPDefs** method. This method, available on both the **sWorkQManager** and **vAWorkQ** objects, returns **vCDQPDef** objects, which provide the CDQP definitions. If called from **sWorkQManager**, it sends a message to the server to retrieve an array of **vCDQPDef** objects, one for each CDQP field defined on the specified work queue. If called from **vAWorkQ**, this returns an array of **vCDQPDef** objects, one for each CDQP field defined on the work queue represented by the local Value Object.



Once you have retrieved a work item Value Object (**vWorkItem**), the CDQP fields that are being used in that work item are available with the **getCDQPs** method. This method returns an array of **vCDQP** objects, one for each CDQP field that is being used in the work item. The **vCDQP** objects provide access to the values in the CDQP fields.



Your filter expressions can include any of the CDQP fields that have been defined on the work queue. For example, assuming LOAN_AMT is listed as one of the CDQP fields for the work queue, the following is a valid filter expression:

```
"LOAN_AMT = 500000"
```

CDQPs Contain Work Item Data

An important thing to understand is that when you filter (or sort) on the values in CDQPs, it's actually "work item data" in the CDQP (as opposed to "case data"). Work item data reflects any "keeps" that have been processed on the work item. In other words, if a user changes the value of a field, then keeps the work item, the CDQP for that field will reflect the changes the user made to the field. The "case data" is only updated when the work item is released.

See "Case Data vs. Work Item Data" on page 166 for more information.

Using Work Queue Parameter Fields

Note - Previous versions of the TIBCO iProcess Objects Server provided "Work Queue Parameter" fields that could be used for filtering and sorting work items based on the value of case data. Work Queue Parameter fields, however, did not provide the flexibility required by some customers. Therefore, a new method using "Case Data Queue Parameter" fields has been implemented (see the previous section). New development should use Case Data Queue Parameter fields to filter on case data instead of the Work Queue Parameter fields (Work Queue Parameter fields will continue to be supported, however).

"Work Queue Parameter" fields allow you to filter work items based on the value of case data fields in your client application. (Work Queue Parameter fields are also used for sorting on case data — see the *Sorting Work Items and Cases* chapter.)

If you have case/field data that you want to filter on (e.g., customer name, loan amount, etc.), it is much more efficient to assign the field value to one of the Work Queue Parameter fields, then filter on that field, instead of directly filtering on the application field. There are four work queue parameter fields available. The default definitions (which can be changed) for these fields are shown below:

Name	Type	Length	Description
SW_QPARAM1	Text	24	WQ Parameter Field 1
SW_QPARAM2	Text	24	WQ Parameter Field 2
SW_QPARAM3	Text	12	WQ Parameter Field 3
SW_QPARAM4	Text	12	WQ Parameter Field 4

These fields can be placed directly in forms, or you can assign the value of an application field to one of the work queue parameter fields through a script. For example:

```
SW_QPARAM1:=LAST_NAME
```

Then, you can filter on the value in the SW_QPARAM1 field. For example, to return only the work items that have a customer last name of Miller, the **FilterExpression** property is set as follows:

```
"SW_QPARAM1?\"Miller\""
```

This would be much more efficient then filtering on the LAST_NAME field.

The **vWorkItem** object contains methods that provide access to the values in the Work Queue Parameter fields — they are **getWorkQParam1 - getWorkQParam4**. These methods return the values you place in the system fields, SW_QPARAM1 - SW_QPARAM4, for each work item.

The **vWorkQ** object contains four methods that return a name for each of the Work Queue Parameter fields — they are **getWorkQParam1Name - getWorkQParam4Name**. If you use the TIBCO iProcess Workspace, these names appear in the column headers if you display the Work Queue Parameter fields in the Work Queue Manager. For information about modifying these names, see the *TIBCO iProcess Workspace (Windows) Manager's Guide*.

Work Queue Parameter Fields vs. Case Data Queue Parameter Fields

Why would you want to use the new Case Data Queue Parameter (CDQP) fields instead of the older Work Queue Parameter fields? The reasons for using each method is shown in the following table.

Case Data Filtering Method	Reasons For Using This Type
Work Queue Parameter Fields	• They are pre-configured, not requiring any administration (where as, CDQP fields require some additional administration). • They are available for all queues, requiring no additional administration. • They are already taking up resources (memory and disk space) whether they are used or not. (Adding four CDQP fields instead of using the already available Work Queue Parameter fields takes up additional resources.) • The load on the Work Item Server is slightly increased for each CDQP. • Configuring CDQP fields requires a TIBCO iProcess Engine shutdown.
Case Data Queue Parameter Fields	The primary reason to use CDQP fields is because if you use the four available Work Queue Parameter fields, then later realize you need more, it will require application changes — with CDQPs, you can just keep adding as many as needed.

Using Regular Expressions

Regular expressions may be included in filter expressions to provide powerful text search capabilities. They can be used when filtering either work items or cases. However, the way in which some regular expression special characters are evaluated differs between work items and cases. See the subsections below for information about the special characters that can be used with regular expressions when filtering work items and cases.

*Note - If using a regular expression when filtering predicted work items (**vPredictedItem** objects), the only special characters that can be used are the asterisk and question mark. They both work as wildcard characters, where the asterisk matches zero or more of any character, and the question mark matches any single character.*

All regular expressions must be in the following format:

constant ? "regular expression"

where:

- **constant** - A constant value or field name. If a field name is included in the expression, the field must be defined as a text data type (SWFieldType = swText). (Note that although the value in DateTime fields (e.g., SW_STARTED) is enclosed in quotes, they cannot be used with regular expressions, as they are not of text data type.)
- **?** - Special character signifying that a regular expression follows (interpreted as an equality operator).
- "**regular expression**" - Any valid regular expression (enclosed in double quotes).

Regular Expressions with Work Item Filtering

The following describes how regular expressions are evaluated when filtering work items (the Work Item Server (WIS) evaluates all work item filter expressions).

Note - If you are moving from a TIBCO iProcess Objects Server that does not have the WIS work item filtering enhancement (CR 12744) to one that does (see page 247 for more information), the way in which some regular expression special characters are evaluated will be different. This can result in a different set of work items being returned using the same filter expression.

A regular expression (RE) specifies a set of character strings. A member of this set of strings is "matched" by the RE. The REs allowed are:

The following one-character REs match a single character.

1. An ordinary character (not one of those discussed in number 2 below) is a one-character RE that matches itself *anywhere* in the constant/field. For example, an RE of “a” will match all constants/fields that contain “a”.
2. A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

   ., *, [, and \ — Period, asterisk, left square bracket, and backslash, respectively. These are always special, except when they appear within square brackets ([ ]; see Item 4 below).

   ^ — Caret or circumflex, which is special at the beginning of an entire RE, or when it immediately follows the left bracket of a pair of square brackets ([ ]) (see Item 4 below).

$ Dollar sign, which is special at the end of an entire RE. The character used to bound (i.e., delimit) an entire RE, which is special for that RE.

3. A period (**.**) is a one-character RE that matches any character except new-line.
4. A one-character RE followed by an asterisk (*) is an RE that matches zero or more occurrences of the one-character RE. If there is any choice, the longest, leftmost string that permits a match is chosen.
5. A non-empty string of characters enclosed in square brackets ([ ]) is a one-character RE that matches any one character in that string, with these additional rules:
   - If the first character of the string is a circumflex (^), the one-character RE matches any character except new-line and the remaining characters in the string. The ^ has this special meaning only if it occurs first in the string.
   - The minus (-) may be used to indicate a range of consecutive characters. For example, [0-9] is equivalent to [0123456789]. The minus sign loses this special meaning if it occurs first (after an initial ^, if any) or last in the string.
   - The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any). For example, [ ]a-f] matches either a right square bracket (]) or one of the ASCII letters a through f, inclusive.
   - The special characters ., *, [, and \ stand for themselves within such a string of characters.

The following rules may be used to construct REs from one-character REs:

1. A one-character RE is an RE that matches whatever the one-character RE matches.
2. The concatenation of REs is an RE that matches the concatenation of the strings matched by each component of the RE. For example, an RE of “abc” will match all constants/fields that contain “abc” anywhere in the constant/field.

An entire RE may be constrained to match only an initial segment or final segment of a line (or both):

1. A circumflex (^) at the beginning of an entire RE constrains that RE to match an initial segment of a line.
2. A dollar sign ($) at the end of an entire RE constrains that RE to match a final segment of a line.
3. The construction ^*entire RE*$ constrains the entire RE to match the entire line.

Regular Expressions with Case Filtering

Regular expressions can be used when filtering cases, however, the only special characters that can be used are the asterisk and question mark. They both work as wildcard characters, where the asterisk matches zero or more of any character, and the question mark matches any single character.

The reason only the asterisk and question mark can be used when filtering cases is that cases are filtered by the database, which cannot interpret the other special characters.

If you are moving from a TIBCO iProcess Objects Server that does not have the database case filtering enhancement (CR 13182) to one that does (see *page 247* *for more information), the way in which some regular expression special characters are evaluated will be different. This can result in a different set of cases being returned using the same filter expression.*

Using Escape Characters in the Filter Expression

Filter expressions require a string value. Therefore, if within the string value, you are required to provide another string, you must use an escape character to provide the quoted string within a string.

Use the back slash to indicate that the next character is a special character. In the example below, the back slashes indicate that the quotes that follow them are quoting the string "LOAN", and are not the ending quotes for the filter expression string.

```
"SW_PRONAME=\"LOAN\""
```

Filtering on Empty Fields

To filter on an empty field, you can use either of the following:

- compare the field with SW_NA, which checks to see if the field is "not assigned." For example:

  ```
  "SOC_SEC_NUM=SW_NA"
  ```

- compare the field to an empty set of quotes. For example:

  ```
  "SOC_SEC_NUM=\"\""
  ```

Note - See the previous section for information about using escape characters.

The primary purpose of SW_NA is to determine if fields have been assigned a value. However, it can also be used to determine if system fields have been assigned a value when you are filtering work items (e.g., “SW_CASEDESC=SW_NA”). Note, however, you cannot use SW_NA when filtering cases — the database is not able to interpret it.

How to Specify Ranges of Values

Ranges of values can be specified in your filter expressions. This functionality, however, is limited to filtering on work items only — you cannot use range filtering when filtering cases.

Ranges must use the following format:

```
FilterField=[val1-val2|val3|val4-val5|.....|valn]
```

You can specify multiple ranges or single values, each separated by a vertical bar. The entire range expression is enclosed in square brackets. Only the ‘=’ equality operator is allowed in a range filter expression.

Dates are specified as:

```
!dd/mm/yyyy!
```

*Note - The ordering of the day, month and year is specified in the **staffpms** file (see “Date Format” on page 178).*

Times are specified as:

```
#mm:hh#
```

DateTimes are specified as:

```
"dd/mm/yyyy mm:hh"
```

Range Filter Example 1:

This example returns the work items with case numbers between 50 and 100, and between 125 and 150, as well as the work item with case number 110:

```
SW_CASENUM=[50-100|110|125-150]
```

Range Filter Example 2:

To return all work items that arrived in the queue between 09/01/2000 and 09/03/2000 (inclusive), and that have a priority equal to 50:

```
SW_ARRIVALDATE=[!09/01/2000! - !09/03/2000!] AND SW_PRIORITY=50
```

Closing/Purging Cases Based on Filter Criteria

The **sCaseManager** object contains methods that allow you to close or purge cases based on filter criteria. These methods are:

- **closeCasesByCriteria** - This method closes cases that match the specified filter criteria. You must have case administration privilege to close a case (defined in TIBCO Business Studio). You also cannot close a case from a slave node.
- **purgeCasesByCriteria** - This method purges cases that match the specified filter criteria. You must have case administration privilege to purge a case (defined in TIBCO Business Studio). You also cannot purge a case from a slave node.

Both of these methods require a parameter that specifies a filter string expression. Use the filter expression syntax described in this chapter.

Closing and purging cases require that the user have system administrator authority (MENUNAME = ADMIN). See "User Attributes" on page 187 for information about the MENUNAME attribute.

Default Filter Criteria

The TIBCO iProcess Server Objects provide the ability to set *default* work item filter and sort criteria for a work queue.

Default criteria is a feature of the TIBCO iProcess Objects Server that allows you to save a specific criteria that persists for the work queue. Note, however, that the default criteria is not automatically applied to a pageable list of work items that is created for that work queue. Filter criteria must *always* be passed in the form of a **vWICriteria** Value Object when the list is requested. To apply the default criteria, you must call the **getDefaultCriteria** method (which returns a **vWICriteria** object representing the default criteria), then pass the **vWICriteria** object to the **getWorkItemList** or **getAWorkItemList** method when requesting the pageable list.

Note - If you use the TIBCO iProcess Workspace (Windows), filter criteria that are defined on the Work Queue Manager Work Item List Filter dialog become the default filter criteria for that work queue. The default filter criteria defined on this dialog can be viewed and/or affected by the methods described below.

The **sWorkQ** object contains methods that allow you to affect the default filter criteria (note that at the same time these methods are affecting the default sort criteria for the work queue — see "Setting Default Sort Criteria" on page 279):

- **changeDefaultCriteria** - This method sets the default criteria for the work queue based on the **vWICriteria** object passed in the method call. These filter criteria will persist on this work queue until changed again with this method or cleared with the **clearDefaultCriteria** method. (Note that this method is also setting the default sort criteria based on sort fields that are included in the **vWICriteria** object passed in the method call.)
- **clearDefaultCriteria** - This method clears the default filter criteria that were set either through the Work Queue Manager or by using the **changeDefaultCriteria** method. (Note that this also clears any default sort criteria that have been defined.)
- **getDefaultCriteria** - This method returns a **vWICriteria** object, indicating the currently set default criteria for the work queue. To apply default criteria, you must call this method to obtain the **vWICriteria** object, then pass that Value Object with the **getWorkItemList** or **getAWorkItemList** method when requesting the pageable list of work items.

You can **only** persist filter criteria that are a subset of those supported by the Work Queue Manager or an exception will be thrown when you execute **changeDefaultCriteria**. The following are the filter criteria that are supported by the Work Queue Manager.

System Field	Description
SW_ARRIVAL	Arrival date and time
SW_ARRIVALTIME	Arrival time
SW_ARRIVALDATE	Arrival date
SW_CASEDESC	Case description
SW_CASENUM	Case number
SW_CASEREF	Case reference number
SW_DEADLINE	Deadline date and time
SW_DEADLINETIME	Deadline time
SW_DEADLINEDATE	Deadline date
SW_EXPIRED	Deadline Expired Flag
SW_FWDABLE	Forwardable Items
SW_HASDEADLINE	Deadline Set Flag
SW_HOSTNAME	Host Name
SW_NEW	Unopened Work Item Flag
SW_PRIORITY	Priority of work item
SW_PRODESC	Procedure Description
SW_PRONAME	Procedure Name
SW_QPARAM1	Work Queue Parameter1
SW_QPARAM2	Work Queue Parameter2

System Field	Description
SW_QPARAM3	Work Queue Parameter3
SW_QPARAM4	Work Queue Parameter4
SW_RELABLE	Releasable Work Item Flag
SW_STEPDESC	Form (Step) Description
SW_STEPNAME	Form (Step) Name
SW_URGENT	Urgent Work Item Flag

Also note the when using the **changeDefaultCriteria** method, your filter expressions must conform to the following guidelines:

- The *only* equality operator you can use is ‘=’.
- You cannot use any of the following equality operators: ‘?’,’<‘, ‘<=’, ‘>’, ‘>=’, and ‘<>’.
- You cannot use the OR logical operator.
- And since ‘?’ is not allowed, no regular expression syntax can be used.

16

Sorting Work Items and Cases

Introduction

The TIBCO iProcess Server Objects provide the ability to *sort* work items and cases so they can be presented to the user in a desired order. For example, you may want all work items in the work queue sorted by priority (SW_PRIORITY), in ascending order.

You can also sort using multiple criteria. For example, you may want all work items that are flagged as urgent (SW_URGENT) to be listed first in the queue, and also sort the same work items in ascending order according to the date and time they arrived in the work queue (SW_ARRIVED).

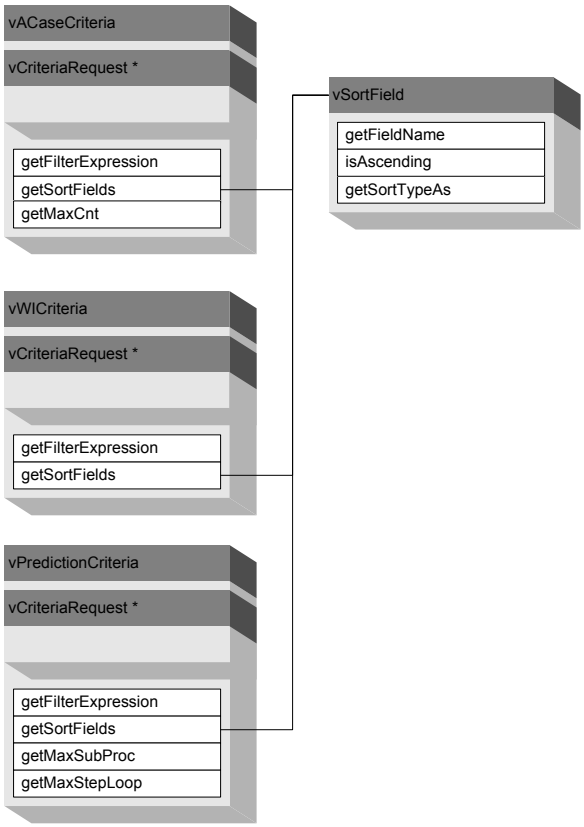
Specifying Sort Criteria

Three "criteria" objects are used to specify sort criteria:

- **vWICriteria** (for work items)
- **vACaseCriteria** (for cases)
- **vPredictionCriteria** (for predicted work items)

Note that these criteria objects are used for both sorting and filtering work items and cases — see the appropriate *Filtering Work Items and Cases* chapter on page 193, page 221, or page 247 for information about filtering.

Work items and cases that can be sorted are always returned in pageable lists (**sPageableList**, **sPageableListR**, or **sPageableListJ** objects) — see "Working with Lists" on page 137 for information about using pageable lists after the sorted work items or cases have been returned from the server.



The methods that return sorted work items and cases are summarized in the table below.

Method	Uses this Criteria Object	Returns Pageable List of this Object
sWorkQ.getWorkItemList	vWICriteria	vWorkItem
sWorkQ.getAWorkItemList	vWICriteria	vAWorkItem
sCaseManager.getACaseList	vACaseCriteria	vACase
sCaseManager.getPredictedItemList	vPredictionCriteria	vPredictedItem

To specify sort criteria for a pageable list of work items or cases, follow these steps:

1. Construct **vSortField** objects, one for each field on which you want to sort the work items or cases. The **vSortField** objects can specify either of the following types of fields:
   - **System Fields** - These are symbolic references to information about the case or work item. For example, its priority (SW_PRIORITY), the case number (SW_CASENUM), etc. See "System Fields used in Sorting" on page 275 for more information.
   - **Case Data Fields** - These are the fields displayed on the form. You can sort according to the value of these fields. See "Sorting on Case Data Fields" on page 277 for more information.

   Note - Field names that are added to the SortFields list that are not valid system fields or case data fields are silently ignored.
2. Construct a **vACaseCriteria**, **vWICriteria**, or **vPredictionCriteria** object, passing an array of the **vSortField** objects constructed in step 1, which specify the fields on which the cases or work items are to be sorted.
3. Pass the **vACaseCriteria**, **vWICriteria**, or **vPredictionCriteria** object as an input parameter with one of the methods listed in the table above, depending on the type of object you want returned in the pageable list.

Sorting in an Efficient Manner

Sorting and filtering take place in a single operation — the Work Item Server (WIS), TIBCO iProcess Objects Server, or database (whichever is performing the filter/sort operation) evaluates the filter expression to obtain a result set, then that result set is sorted.

To perform this filter/sort operation in an efficient manner, there are a number of things you need to be concerned with. These include things such as whether the server needs to "get case data". The efficiency of the filter/sort operation can be greatly affected by these. See the *Filtering Work Items and Cases* chapter that is applicable to you (depending on the filtering enhancements in your TIBCO iProcess Objects Server) for more information about how to ensure an efficient filter/sort operation. These chapters are on page 193, page 221, and page 247.

System Fields used in Sorting

The built-in system fields (e.g., SW_CASE, SW_STARTER, etc.) provide the TIBCO iProcess Engine with references to information about work items and cases. These fields are primarily used by the TIBCO iProcess Engine (specifically, the Work Item Server) when performing filtering and sorting functions.

The information that is available to the TIBCO iProcess Engine through the system fields is also available to the application through methods on various Value Objects. For example, **SW_CASENUM** is available to the client with the **vCaseId.getCaseNumber** method, **SW_PRIORITY** is available to the client with the **vWorkItem.getPriority** method, etc. The TIBCO iProcess Engine, however, doesn't have access to the Value Objects, so those methods can't be used to specify sort criteria — instead, the system field names need to be used.

The system fields that are available for sorting are listed in the table below. Note that some system fields are only applicable to sorting on work items, some only for sorting on cases, and some are applicable to both (see the "Applies to" columns).

The "WIS-compatible" column tells you if the Work Item Server (WIS) can process that particular system field. This is applicable only when sorting work items (cases are always sorted by either the TIBCO iProcess Objects Server or the database). See "Can the WIS Perform the Sort Operation?" on page 201 and page 228 for more information.

Sort Criteria	System Field	Data Type	Length	WIS-compatible	Applies to work items	Applies to cases
Arrival date and time	SW_ARRIVAL	DateTime	16	X	X	
Case description	SW_CASEDESC	Text	24	X	X	X
Case ID in procedure	SW_CASEID	Numeric	7		X	
Case number	SW_CASENUM	Numeric	15	X	X	X
Case reference number	SW_CASEREF	Text	20		X	X
Deadline date and time	SW_DEADLINE	DateTime	16	X	X	
Deadline time	SW_DEADLINETIME[a]	Time	5	X	X	
Deadline expired flag (1 - expired; 0 - does not have expired deadline)	SW_EXPIRED	Numeric	1	X	X	
Deadline set flag (1 - has deadline; 0 - no deadline)	SW_HASDEADLINE	Numeric	1	X	X	
Forwardable work item flag (1 - forwardable; 0 - not forwardable)	SW_FWDABLE	Numeric	1	X	X	
Host name	SW_HOSTNAME	Text	24 or 8[b]	X	X	
Locker of the work item (username)	SW_LOCKER[a]	Text	24 or 8[b]	X[c]	X	
Mail ID	SW_MAILID	String or Numeric [d]	7 (integer) 45 (string)		X	

Sort Criteria	System Field	Data Type	Length	WIS-compatible	Applies to work items	Applies to cases
Outstanding work item count (not available on TIBCO iProcess Engines)	SW_OUTSTANDCNT	Numeric	7			X
Pack file (not available on TIBCO iProcess Engines)	SW_PACKFILE	Text	13		X	
Priority of work item	SW_PRIORITY	Numeric	3	X	X	
Procedure description	SW_PRODESC	Text	24	X	X	X
Procedure name	SW_PRONAME	Text	8	X	X	X
Procedure number	SW_PRONUM	Numeric	7		X	X
Releasable work item (no input fields) (1 - releasable; 0 - not releaseable)	SW_RELABLE	Numeric	1	X	X	
Started date and time of the case	SW_STARTED	DateTime	16			X
Starter of the case (username@node)	SW_STARTER[a]	Text	24 or 8[b]		X	X
Status of the case (“A” - active; “C” - closed)	SW_STATUS	Text	1			X
Step (work item) description	SW_STEPDESC	Text	24	X	X	
Step (work item) name	SW_STEPNAME	Text	8	X	X	
Step (work item) number in procedure	SW_STEPNUM	Numeric	7		X	
Terminated date and time of the case	SW_TERMINATED	DateTime	16			X
Unopened work item (1 - unopened; 0 - has been opened)	SW_NEW	Numeric	1	X	X	
Urgent flag (1- urgent; 0 - not flagged as urgent)	SW_URGENT	Numeric	1	X	X	
Work queue parameter 1	SW_QPARAM1	Text	24	X	X	
Work queue parameter 2	SW_QPARAM2	Text	24	X	X	
Work queue parameter 3	SW_QPARAM3	Text	12	X	X	
Work queue parameter 4	SW_QPARAM4	Text	12	X	X	

a. SW_DEADLINETIME, SW_LOCKER, and SW_STARTER are inherently inefficient — they may slow the sort process.
b. This has a length of 24 for long-name systems, or 8 for short-name systems.
c. SW_LOCKER is WIS-compatible only if your TIBCO iProcess Objects Server has implemented CR 13397.
d. If using a TIBCO Process Engine, SW_MAILID is an integer of length 7; if using a TIBCO iProcess Engine, SW_MAILID is a string of length 45.

Sorting on Case Data Fields

You can sort **cases** using any case data field in the case. Note, however, that the values in these fields contain "case data" (as opposed to "work item data"). "Case data" is updated with the values that have been entered into the fields of a work item only when you "release" that work item. For more information about case data and work item data, see "Case Data vs. Work Item Data" on page 166.

You can sort **work items** using case data fields only if they have been designated as one of the following:

- **Case Data Queue Parameter (CDQP) Fields** - CDQP fields are a more recent addition than Work Queue Parameter fields (see below) that allow you to filter and/or sort on an unlimited number of case data fields that appear in work items on your work queue.
- **Work Queue Parameter Fields** - These fields are used by assigning a case data field value to one of the pre-defined work queue parameter fields, then using the Work Queue Parameter field in filter or sort criteria. These fields have been superseded by CDQP fields as they were considered too limiting since there are only four of them.

These are described in the following subsections.

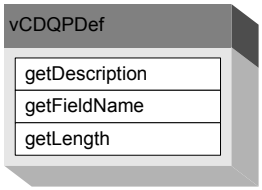
Using Case Data Queue Parameter Fields

Case Data Queue Parameter (CDQP) fields provide an efficient method of sorting on the value of fields in your client application. To make use of this functionality, you must first pre-designate the fields as fields that can be sorted on — once designated, these fields are called "Case Data Queue Parameter" fields. (The TIBCO iProcess Objects Server will still sort on case data fields that have not been designated as Case Data Queue Parameter fields, but it will be much less efficient.)

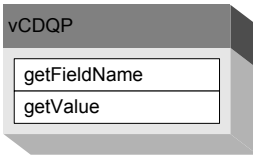
Note - Case Data Queue Parameter fields are also used for efficiently filtering on case data. See the appropriate Filtering Work Items and Cases chapter on page 193, page 221, or page 247.

Fields are designated as Case Data Queue Parameter fields with the utility, **swutil**. This utility is used to create a list, on the TIBCO iProcess Engine, of the case data fields that are available to use for sorting. For information about using **swutil**, see the *TIBCO iProcess swutil and swbatch Reference Guide*.

Once you have created the list of CDQP fields with **swutil**, this list of fields is available with the **getCDQPDefs** method. This method, available on both the **sWorkQManager** and **vAWorkQ** objects, returns **vCDQPDef** objects, which provide the CDQP definitions. If called from **sWorkQManager**, it sends a message to the server to retrieve an array of **vCDQPDef** objects, one for each CDQP field defined on the specified work queue. If called from **vAWorkQ**, this returns an array of **vCDQPDef** objects, one for each CDQP field defined on the work queue represented by the local Value Object.



Once you have retrieved a work item Value Object (**vWorkItem**), the CDQP fields that are being used in that work item are available with the **getCDQPs** method. This method returns an array of **vCDQP** objects, one for each CDQP field that is being used in the work item. The **vCDQP** objects provide access to the values in the CDQP fields.



When constructing the **vSortField** objects that will be used in your criteria object (**vWICriteria**), you will use the CDQP field name for the *aFieldName* parameter.

Using Work Queue Parameter Fields

Note - Previous versions of the TIBCO iProcess Objects Server provided "Work Queue Parameter" fields, which could be used for filtering and sorting work items based on the value of case data in the Work Queue Parameter fields. Work Queue Parameter fields, however, did not provide the flexibility required by some customers. Therefore, a new method using "Case Data Queue Parameter" fields has been implemented in the TIBCO iProcess Objects Server (see the previous section). New development should use Case Data Queue Parameter fields to sort on case data instead of the Work Queue Parameter fields (Work Queue Parameter fields will continue to be supported, however).

"Work Queue Parameter" fields allow you to sort work items based on the value of case data fields in your client application. (Work Queue Parameter fields are also used for filtering on case data — see the appropriate *Filtering Work Items and Cases* chapter on page 193, page 221, or page 247.)

If you have case data you want to sort on (e.g., customer name, loan amount, etc.), it is much more efficient to assign the field value to one of the Work Queue Parameter fields, then sort on that field, instead of directly sorting on the application field. There are four work queue parameter fields available:

Name	Type	Length	Description
SW_QPARAM1	Text	24	WQ Parameter Field 1
SW_QPARAM2	Text	24	WQ Parameter Field 2
SW_QPARAM3	Text	12	WQ Parameter Field 3
SW_QPARAM4	Text	12	WQ Parameter Field 4

These fields can be placed directly in forms, or you can assign the value of an application field to one of the work queue parameter fields through a script. For example:

```
SW_QPARAM1:=LAST_NAME
```

Your application would use the Work Queue Parameter field name when constructing a **vSortField** object that will be used when constructing the criteria object (**vWICriteria**). This would be much more efficient than sorting directly on the LAST_NAME field.

The **vWorkItem** object contains methods that provide access to the values in the Work Queue Parameter fields — they are **getWorkQParam1 - getWorkQParam4**. These methods return the values you place in the system fields, SW_QPARAM1 - SW_QPARAM4, for each work item.

The **vWorkQ** object contains four methods that return a name for each of the Work Queue Parameter fields — they are **getWorkQParam1Name - getWorkQParam4Name**. If you use the TIBCO iProcess Workspace (Windows), these names appear in the column headers if you display the Work Queue Parameter fields in the Work Queue Manager. For information about modifying these names, see the *TIBCO iProcess Workspace (Windows) Manager's Guide*.

Setting Default Sort Criteria

The TIBCO iProcess Server Objects provide the ability to set *default* work item filter and sort criteria for a work queue.

Default criteria is a feature of the TIBCO iProcess Objects Server that allows you to save a specific criteria that persists for the work queue. Note, however, that the default criteria is not automatically applied to a pageable list of work items that is created for that work queue. Filter criteria must always be passed in the form of a **vWICriteria** Value Object when the list is requested. To apply the default criteria, you must call the **getDefaultCriteria** method (which returns a **vWICriteria** object representing the default criteria), then pass the **vWICriteria** object to the **getWorkItemList** or **getAWorkItemList** method when requesting the pageable list.

Note - If you use the TIBCO iProcess Workspace (Windows), sort criteria that are defined on the Work Queue Sort Criteria dialog become the default sort criteria for that work queue. The default sort criteria defined on this dialog can be viewed and/or affected by the methods described below.

The **sWorkQ** object contains methods that allow you to affect the default sort criteria (note that at the same time these methods are affecting the default filter criteria for the work queue — see the "Default Filter Criteria" section in the appropriate "Filtering Work Items and Cases" chapter on page 219, page 245, or page 270):

- **changeDefaultCriteria** - This method sets the default sort criteria for the work queue based on the **vWICriteria** object passed in the method call. This sort criteria will persist on this work queue until changed again with this method or cleared with the **clearDefaultCriteria** method. (Note that this method is also setting the default filter criteria based on the filter expression that is included in the **vWICriteria** object passed in the method call.)
- **clearDefaultCriteria** - This method clears the default sort criteria that were set either through the Work Queue Manager or by using the **changeDefaultCriteria** method. (Note that this also clears any default filter criteria that have been defined.)
- **getDefaultCriteria** - This method returns a **vWICriteria** object, indicating the currently set default criteria for the work queue. To apply default criteria, you must call this method to obtain the **vWICriteria** object, then pass that Value Object with the **getWorkItemList** or **getAWorkItemList** method when requesting the pageable list of work items.

You can **only** persist sort criteria that are a subset of those supported by the Work Queue Manager or an exception will be thrown when you execute **changeDefaultCriteria**.

The following are the sort criteria that are supported by the Work Queue Manager. These criteria can be persisted with the **changeDefaultCriteria** method:

System Field	Description
SW_ARRIVAL	Arrival date and time
SW_CASEDESC	Case description
SW_CASENUM	Case number
SW_CASEREF	Case reference number
SW_DEADLINE	Deadline date and time
SW_EXPIRED	Deadline Expired Flag
SW_FWDABLE	Forwardable Items
SW_HASDEADLINE	Deadline Set Flag

System Field	Description
SW_HOSTNAME	Host Name
SW_NEW	Unopened Work Item Flag
SW_PRIORITY	Priority of work item
SW_PRODESC	Procedure Description
SW_PRONAME	Procedure Name
SW_QPARAM1	Work Queue Parameter1
SW_QPARAM2	Work Queue Parameter2
SW_QPARAM3	Work Queue Parameter3
SW_QPARAM4	Work Queue Parameter4
SW_RELABLE	Releasable Work Item Flag
SW_STEPDESC	Form (Step) Description
SW_STEPNAME	Form (Step) Name
SW_URGENT	Urgent Work Item Flag

Sorting as a Specified Data Type

When constructing **vSortField** objects, the *aSortTypeAs* parameter allows you to specify that the sort fields be converted to the specified data type before the sort comparison is performed.

When sorting **work items**, sorting by a specified type is only applicable when sorting on:

- Work Queue Parameter fields (SW_QPARAM1-4) - see "Using Work Queue Parameter Fields" on page 278
- Case Data Queue Parameter fields - see "Using Case Data Queue Parameter Fields" on page 277
- Case Description (SW_CASEDESC)

When sorting **cases**, you can sort on a specified data type for *any* field.

The data types that can be specified with the *aSortTypeAs* parameter are enumerated in **SWSortType**.

SWSortType		
Constant	**Description**	**Value**
swDateSort	Sort as date	'D'
swDateTimeSort	Sort as date/time	'B'
swNumericSort	Sort as real number	'R'
swTextSort	Sort as text	'A'
swTimeSort	Sort as time	'T'

The TIBCO iProcess Objects Server will convert the value of the sort field to the specified sort type before doing the sorting. For example, text fields containing numeric information could be sorted as numbers by setting the sort type accordingly. Note, however, that if the sort field does not contain something readily convertible to the specified type, the sort results may be unexpected. For example, if sorting text as a numeric field but some of the text fields contain non-numeric data, the results of the conversion are not defined, so the sort results may not be what you expected.

17

Error Handling

Introduction

Many of the methods on the Server Objects take arrays as parameters. This allows you to submit a group of items for processing rather than having to make multiple calls with each call referencing a single item. For example:

```
vWIFieldGroup[] lockItems(String[] aWorkItemTags,
                          vWIFGContent aWIFGContent)
                               throws vException, vExWIFieldGroup
```

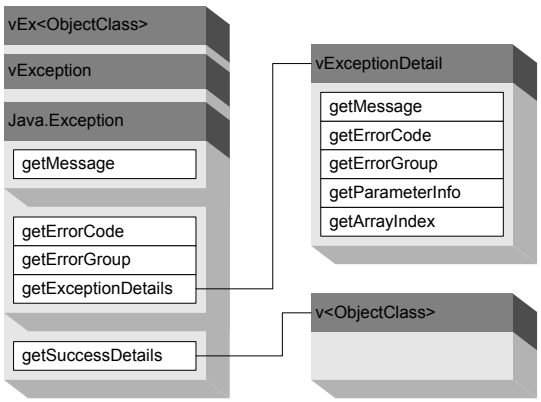
The **lockItems** method accepts an array of work item tags, identifying the work items to process. The method attempts to lock all work items referenced in the array.

It is possible that the server will be able to lock some of the items, but fail to lock others. For example, some of the work items might already be locked or perhaps an invalid tag is passed in. The error handling is designed to address those instances where multiple items are submitted for processing and some succeed but others fail.

Methods on the TIBCO iProcess Server Objects will throw an error if they are unable to process ALL of the work requested, but at the same time, as much work as possible will be completed. This is especially important with functions like locking work items or creating users, where we may have already processed a good portion of the items when one signals an error. We do not want to have to back out of the valid work we have already done.

When all processing completes normally, the method returns any results as an array. The problem is that if we throw an error, then we cannot also return a result to the method. If an error is thrown, the result value is not assigned to the client variable.

To provide the capability to throw an error and also return valid data, we use a custom exception class (shown in the illustration). This class has the capability of returning two arrays, one for items that succeed and one for items that generated errors.



This custom exception class is derived from the **Java.Exception** object. The use of this class is described below.

There are three levels of errors that may occur on methods:

- The **first level of error** is generated by the method as a whole — that is, it was generated for a reason other than parameters that could not be processed. For example:

```
vProcId[] getStartProcIds()
                          throws vException
```

Since this method does not have parameters, if it fails, it will throw a **vException** error. Note that even methods that include parameters may fail with a **vException** if it fails for a reason other than an invalid parameter.

The **vException** object provides **getErrorCode** and **getErrorGroup** methods to determine what caused the error. The **getExceptionDetails** method returns a zero-length array for this type of error.

- The **second level of error** is generated by methods that pass in parameter arrays but do not expect data to be returned. For example:

```
void changeAttributes(vAttribute[] aChangedAttributes)
                              throws vException
```

This level of error occurs when one or more of the parameters being passed in fails to process correctly. It causes a **vException** to be thrown. Unlike the first level of error, however, this level also provides details about the exception in the form of **vExceptionDetail** objects. One **vExceptionDetail** object is added to the **getExceptionDetails** array for each parameter that failed to process properly.

The **vExceptionDetail** object provides details about each failure that occurred, including the message returned from the server (**getMessage**). The **getParameterInfo** method returns information that can be used to help determine the cause of the failure. Depending on the particular method that caused the exception, the **getParameterInfo** method may return the value of the parameter that failed, or a property that helps identify an object that was passed in as a parameter that failed (typically, the name of the object). The **getArrayIndex** method provides an index value into the array that caused the error. This can be used to pinpoint the parameter in the input array.

- The **third level of error** is generated on methods where you are passing in parameters to be processed, and that return an array of data in response to the method call. For example:

```
vProc[] getProcs(String[] aProcTags)
                     throws vException, vExProc
```

If all items process successfully, an array of results is returned from the call. If any of the items fail to process successfully, a **vExProc** error is thrown (the generic name of this error exception is **vEx<Object Class>**, where <Object Class> is the object class being returned by the method call).

If a **vEx**<ObjectClass> exception is thrown, details of the exception are provided in the **getExceptionDetails** array (exactly as in the second level of error as described above).

If a **vEx**<ObjectClass> exception is thrown, results are not returned to the call. For every item that succeeded, the results are returned on the **getSuccessDetails** array, one **v**<ObjectClass> for each successful object. These are of the same type that would have been returned by the method call.

Since we want to be able to process the method results and the **getSuccessDetails** in the same way, both arrays need to look the same to the calling program. Because of strong type checking, there is a unique error object for every type of object that may be returned by a method, i.e., the **vEx**<ObjectClass> is thrown for methods that return arrays of type <ObjectClass>. For example, **getAttributes** returns an array of **vAttribute** objects; if one of the attributes requested is invalid, **getAttributes** will throw a **vExAttribute** error and the **getSuccessDetails** array will return an array of **vAttributes** that processed correctly.

18

Client Configuration

This chapter provides information about the following items that can be configured in the client:

- **Client Log** - This log records messages generated by base objects. For more information, see "Client Log" on page 283.
- **Message Wait Time** - This specifies the amount of time the client will wait for a response from the server. For more information, see "Message Wait Time" on page 292.
- **Encoding Using ICU Conversion Libraries** - ICU conversion libraries can be used to specify the desired character encoding. For more information, see "Character Encoding Using ICU Conversion Libraries" on page 293.

Client Log

The client log records messages generated from base objects. These messages are useful for debugging purposes. Typically, this log is used by development engineers to assist in debugging a customer's client application. Only ONE client log exists per JVM process.

Note that the following logs are also available:

- **TIBCO iProcess Objects Server Log** - This log records messages generated by the TIBCO iProcess Objects Server. The server log can be configured and reset using **sNodeManager**. For more information, see the *TIBCO iProcess Objects Server Administrator's Guide*.
- **Audit Log** - This log records information about administrative functions that are performed (e.g., adding/removing users, changing passwords, etc.). For more information, see the *TIBCO iProcess Objects Server Administrator's Guide*.
- **UNIX System Log** - This log records activity performed on UNIX systems. For more information, see the *TIBCO iProcess Objects Server Administrator's Guide*.

Refer to the appropriate document for information about the other logs listed above.

Client Log Overview

When TIBCO iProcess Server Objects attempt to log a message, if the log exists, the new message is appended to the existing log. If the log does not exist, a new log is created, then a header is written to the log, followed by the message.

By default, only swLogError-level messages are written to the log. These are errors not expected by the client application. So if no errors are occurring, no log is created.

The client log contains one line for each message, in the following format:

```
f|pppppppp|tttttttt|hhhhhhhh|dd/dd/dddd dd:dd:dd.ddd|cccccccc|mmmmmmmm|llll|UserMsg
```

where:

f	Line format (B - Basic, M - Contains memory values)
pppppppp	Process ID
tttttttt	Thread ID
hhhhhhhh	Hood ID, where: 00000000 = Msg from sClientLog 00000001 = Msg from receive thread 00000002 = (not used) 00000003 = (not used) 00000004 = JNI boundary error 00000005 = Timer thread usage 00000006 = Shared function (usually swrtns.cpp) 00000009 = Undefined hood 0000000A & above = Logged into new user session (new Server Object created)
dd/dd/dddd dd:dd:dd.ddd	Date and time
cccccccc	Log category (hex format)
llll	Log level (ERR, WARN, INFO, DEBG)
mmmmmmmm	Message type (see **SWLogMessageType** on page 290)
UserMsg	User message

Hood ID

Log messages that have a common source or a natural affinity are said to be from a common hood. The Hood ID identifies that relationship. For example, all log messages generated in the process of receiving TIBCO iProcess Objects Server messages are from the receive thread hood (00000001), and all messages generated by the client log code are from the sClientLog hood (00000000).

Logging on to a server generates a new Hood ID. All objects in the hierarchy below this node are in the same hood (0000000B and above).

User Message

The user message is the text that comes out of the object when the log message is generated. The user message should adhere to the following format (adhering to this format is important for the log to be properly read by the Client Log reader program):

```
<Module/Class>.<Routine>:<Text Message> (Relevant Data)
```

where:

Module/Class	The source module or class file from which the message originates.
"."	A period separator.
Routine	The specific property, method or routine in the Module/Class.
":"	A colon separator.
Text Message	Generic text message (should not contain instance or error-specific data).

(Relevant Data)	Instance or error-specific data. This is the only place where you can insert C-language format specifications or values of variables into the message. This data is enclosed in parentheses.

An example message written to the client log is shown below:

```
B|00000076|0000015C|00000001|06/05/2003 08:27:43.752|7FFFFFFF|ERR |RcvThread.main: Socket Problem on Read,(WSAGetLastError = 10054, sockfd = 568)
```

Note - These messages are always logged as swLogInformation (log level) and swCatSEOUser (category). So you must ensure this level and category are enabled.

Controlling the Client Log

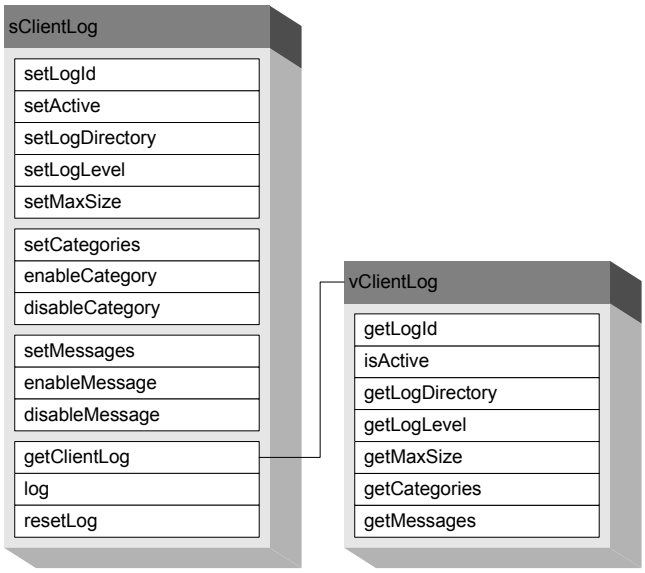
The client log can be programmatically controlled. For instance, you can turn the log on or off, specify the location of the log, specify the categories of messages that will be written to the log, etc. Typically, these settings are modified only upon the request of a TIBCO engineer. The log can be controlled using the following methods:

- using the **sClientLog** Server Object (dynamic)
- through **Registry** settings (Windows only) (read when the JVM process is started)
- through **environment variables** (UNIX only) (read when the JVM process is started)

The sClientLog Object

The **sClientLog** Server Object provides methods that allow configuration and control of the client log from the client application. It allows you to define the location of the log file, the level of logging, the size of the log, etc. It also contains the **getClientLog** method that allows you to retrieve a **vClientLog** Value Object, which provides access to the current log file settings.

The controls provided through the methods on the sClientLog object are described in the subsections that follow. Note - If multiple clients modify the log setting, the last modification is the one that takes effect.



Registry Settings (Windows only)

When the client log is initially created, it attempts to obtain its property values from the Registry. If the Registry entries exist, those values are used. If the Registry entries do not exist, a Registry key is created at the path shown below.

```
HKEY_CURRENT_USER
    SOFTWARE
        Staffware plc
            Staffware SSO Client
                Logs
                    SWClient_processname
```

Note, however, that the Registry path is slightly different if running from IIS. Instead of HKEY_CURRENT_USER, the Registry location is HKEY_USERS\.DEFAULT. The ".DEFAULT" portion of the Registry path when running IIS is used after initial install. However, if IIS is upgraded, that portion of the Registry path may change to a Security Identifier (SID) for the NT AUTHORITY\Network Service user. Microsoft TechNet provides a tool called **psgetsid** that can be used to determine a user's SID. It can be found here: http://technet.microsoft.com/en-us/sysinternals/bb897417.aspx. An example of using psgetsid:

```
C:\psTools\psgetsid "NT AUTHORITY\Network Service"
```

Use the SID returned by psgetsid in the Registry path.

Note - If you install 32-bit software on a 64-bit machine, the Registry path will include "Wow6432Node" as follows:

```
HKEY_CURRENT_USER\SOFTWARE\Wow6432Node\Staffware plc\...
```

When the Registry entries are created, the following default values are written into the Registry:

- Active = "1" (true)
- Categories = "7FFFFFF3" (all categories except swCatConstDestr (object constructors/destructors) and swCatReceiveThread — if you need this information in the log, you can use the setCategories method to set it to swCatAll, or use the enableCategory method to enable the swCatConstDestr and/or swCatReceiveThread category)
- LogDirectory = [See "Log File Directory" on page 287]
- LogLevel = "1" (Error level — the lowest level)
- MaxSize = "15"
- LogId = "SWClient_*processname*"
- Messages = "7FFFFFFF" (All messages)

Since the client log always tries to obtain its property values from the Registry upon creation, you can use the Registry to configure and control the client log by setting the Registry values prior to log creation. You can also prevent a client log file from ever being created by ensuring the entries have been added to the Registry, then setting the **Active** flag to "0" (false).

Environment Variables (UNIX Only)

When the client log is initially created, it attempts to obtain its property values from the environment variables listed below. If the environment variables exist, those values are used. If the environment variables do not exist, the defaults shown in "Registry Settings" above are used (the default directory for UNIX is "**/tmp**").

The environment variables that control the client log are NOT automatically created. If you want to use them to control the log, you must manually create them.

Since the client log always tries to obtain its property values from the environment variables upon creation, you can use them to configure and control the client log by creating them, and setting their values prior to log creation. You can also prevent a client log file from ever being created by ensuring the **Active** flag (SSOClient_Active environment variable) is set to "0" (false).

The following are the UNIX environment variables that can be created to control the client log:

- SSOClient_Active
- SSOClient_Categories

- SSOClient_LogDirectory
- SSOClient_LogLevel
- SSOClient_MaxSize
- SSOClient_LogId
- SSOClient_Messages

Name and Location of the Client Log

Log File Name

The name of the client log on your system defaults to the names shown below, depending on the operating system you are using:

- **Windows** - The name defaults to "**SWClient**_*processname*".
- **UNIX** - The name defaults to "**SSOClient**".

You can change the name of the client log either by using the **sClientLog.setLogId** method, or via the Registry (LogId) on Windows or an Environment Variable (SSOClient_LogId) on UNIX. This name is used in combination with the log file directory (see below), to create the path and filename of the log file in the file system (with a ".log" extension).

Log File Directory

The directory in which the client log is written defaults to the directories shown below, depending on the operating system you are using:

- **Windows -** On Windows, the default directory depends on the version of Windows. For example, on Windows 7, it is:

  `C:\Users\`*`UserName`*`\AppData\Local\Temp\`

  where *UserName* is the logged-in user.

- **UNIX -** `/tmp/`

Note that the *machine* on which the client log file is written will be the machine on which the Server Factory is running.

You can change the directory in which the client log will be saved using the **sClientLog.setLogDirectory** method. Note, however, if you specify a directory with the setLogDirectory method, the directory must exist. If the specified directory does not exist, an entry is written to the log file (in the default directory) concerning the non-existent directory specified in the setLogDirectory method.

Construction of Client Log Name and Directory

The client log file directory/name is constructed in the following way:

```
LogDirectory + ['Slash'] + LogId + ".log"
```

where:

- *LogDirectory* is the path to the directory where the log file will be created. This path must already exist; the logger will not create non-existent directories.
- *Slash* is '\' for Windows and '/' for UNIX. This is appended only if *LogDirectory* was missing a terminating slash. You must supply the correct slashes for the platform in *LogDirectory*; the logger will not correct this if it is incorrect.

- *LogId* is the log filename.
- The extension ".log" is always appended to the *LogId*.

If there is an error when creating or writing to the log file, the logger will always revert to the default path/filename:

- Windows: `C:\Users\UserName\AppData\Local\Temp\SWClient_processname.log`
- UNIX: `/tmp/SSOClient.log`

Also note that if the log directory/filename is valid and the logger is successfully writing to it, then you use the logger API to change the directory/filename to an incorrect value, the logger reverts to the default value (see above), not to the previous correct value you had set; it does not remember the previous correct value.

Activating / Deactivating the Client Log

The **sClientLog.setActive** method specifies whether or not messages are written to the client log. This includes “always log” messages (category swCatAlwaysLog). This method allows you to turn off logging without having to clear the category and message filter settings (which are described in the next section). Calling this method sets the **isActive** flag on the **vClientLog** object. The default setting for isActive is true — logging is enabled.

Setting isActive to False does not inhibit writing the client log properties to the registry; it merely causes all messages to be filtered out.

You can use the isActive flag to prevent the client log from ever being created. Before the client log is created (prior to instantiating a Server Object), set the **Active** entry in the registry to “0” (False). When the client log is created, it will obtain its default values from the registry, including setting the isActive flag to False (based on the setting of the Active entry in the registry).

Filtering the Client Log

The **sClientLog** object contains a number of methods that allow you to specify the categories and types of messages that are written to the client log. The subsections below describe these filtering functions.

Setting the Log Level

The log level is used to filter the amount of information that is written to the client log. You can set the log level using the **sClientLog.setLogLevel** property. The **SWLogLevelType** enumerations can be used to specify the amount of information to record:

SWLogLevelType	Value	Amount of Information
swLogError	1	Least (default)
swLogWarning	2	
swLogInformation	3	
swLogDebug	4	Most

The log levels are hierarchical, from the least amount of information to the most, with each higher level including the information from the levels below it.

Setting the log level to swLogDebug causes ALL categories except swCatConstDestr (object constructors/destructors) to be written to the log (see the next section for information about categories).

Filtering by Category

"Categories" of messages have been defined that allow you to filter according to broad areas of functionality. For example, there are categories that have to do with UDP, WinSock, constructors/destructors, etc. You can filter the client log according to these categories using the **sClientLog.setCategories** method.

The **SWLogCategoryType** enumeration type lists the categories that can be written to the log. Note, however, that you cannot use the enumeration strings to specify categories with the **setCategories** method. It expects an int as a parameter, therefore, you must pass a number. You can combine the values shown in SWLogCategoryType to cause multiple categories to be written to the log.

SWLogCategoryType	Value
swCatAll	0x7FFFFFFF
swCatAlwaysLog	0x00000001
swCatSEOUser	0x00000002
swCatConstDestr	0x00000004
swCatReceiveThread	0x00000008
swCatMessages	0x00000010
swCatMsgSend	0x00000020
swCatMsgReceive	0x00000040
swCatUDP	0x00000080
swCatWinSock	0x00000100
swCatConversion	0x00000200
swCatTiming	0x00000400
swCatMethodCalls	0x00000800
swCatObjectWrapping	0x00001000
swCatMemory	0x00002000

The category setting defaults to 0x7FFFFFF3, which includes all categories except swCatConstDestr (object constructors/destructors) and swCatReceiveThread. If you need object constructor/destructor or receive thread information in the log, you can use the setCategories method to set it to swCatAll, or use the enableCategory method to enable the swCatConstDestr and/or swCatReceiveThread category — see the next section for information about the enableCategory method.

Enabling and Disabling Categories

The sClientLog object provides methods that allow you to enable or disable a single message category. They are:

- **enableCategory(*Category*)** - This method adds the specified category to the list of categories that are written to the client log. The specified category must belong to the **SWLogCategoryType** enumeration.

- **disableCategory(*Category*)** - This method removes the specified category from the list of categories that are written to the client log. The specified category must belong to the **SWLogCategoryType** enumeration. Other enabled categories remain enabled.

Filtering by Message

This functionality allows you to filter messages that are generated when messages are sent to the TIBCO iProcess Objects Server. This is done by using the **sClientLog.setMessages** method. (Note that this is applicable only if the swCatMessages category is enabled; see the previous section.)

The **SWLogMessageType** enumeration type lists the message types that can be written to the log. Note, however, that you cannot use the enumeration strings to specify message types with the **setMessages** method. It expects an int as a parameter, therefore, you must pass a number. You can combine the values shown in SWLogMessageType to cause multiple message types to be written to the log.

SWLogMessageType	Value
swMsgAll	0x7FFFFFFF
swMsgTCP	0x00000001
swMsgUDP	0x00000002
swMsgLogin	0x00000004
swMsgPassword	0x00000008
swMsgUser	0x00000010
swMsgAttribute	0x00000020
swMsgRole	0x00000040
swMsgGroup	0x00000080
swMsgProcedure	0x00000100
swMsgProcedureQuery	0x00000200
swMsgProcedureDefinition	0x00000400
swMsgQueueAccess	0x00000800
swMsgQueueQuery	0x00001000
swMsgCase	0x00002000
swMsgNode	0x00004000
swMsgEvent	0x00008000
swMsgWorkItem	0x00010000
swMsgForwarding	0x00020000
swMsgInstrumentation	0x00040000
swMsgMemoAttachment	0x00080000
swMsgForm	0x00100000
swMsgTable	0x00200000
swMsgListValidation	0x00400000

The default is 0x7FFFFFFF — all message types are written to the log.

Enabling and Disabling Messages

The sClientLog object provides methods that allow you to easily enable or disable a single message type. They are:

- **enableMessage(*Message*) -** This method adds the specified message type to the list of message types that are written to the client log. The specified message type must belong to the **SWLog-MessageType** enumeration.
- **disableMessage(*Message*) -** This method removes the specified message type from the list of message types that are written to the client log. The specified message type must belong to the **SWLogMessageType** enumeration. Other enabled message types remain enabled.

Adding Entries to the Client Log

The **sClientLog.log** method allows you to add text messages to the client log from the customer application. Text message will appear in the *User Message* part of the client log message — see "User Message" on page 284.

User-entered messages are classified as category type *swCatSEOUser* and level type swLogInformation. Therefore, swCatSEOUser and swLogInformation must be enabled for these messages to be written to the log.

Setting the Size of the Client Log

You can specify the maximum size of the client log, in megabytes, using the **sClientLog.setMaxSize** method. When the log exceeds the specified maximum size, it is cleared and restarted (rolled over). A message is written to the log indicating this has occurred.

The default maximum size is 15MB.

Resetting the Client Log

The **sClientLog.resetLog** method can be used to clear the client log. This method causes a flag to be set so that the next time a log record is written, the log file is first cleared and initial header messages are written to the file before the log record is written to the file.

Message Wait Time

You can configure the client so that if a specified period of time elapses waiting for a response from the server, the client will timeout and generate an error. (This is sometimes used when storing memo data — because the TIBCO iProcess Engine must perform file I/O to store memo data, the length of time between when the client requests that a memo value be stored on the server, and the time when the client receives a reply that the data was successfully stored, can be several seconds. Because of this, you may have a desire to configure a message wait time.)

By default, Windows clients timeout in 30 seconds; UNIX clients timeout in 60 seconds. If you want the timeout value to be different, you must configure the message wait time.

To configure the message wait time, you must add a Registry DWORD value (Windows) or environment variable (UNIX), and set it to the number of milliseconds you would like the client to wait before timing out, as described in this section.

The following is the Registry DWORD value that must be added if using Windows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Staffware plc\Staffware SSO Client\MessageWaitTime
```

Note - If you install 32-bit software on a 64-bit machine, the Registry path will include "Wow6432Node" as follows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Staffware plc\...
```

The following is the environment variable that must be added if using UNIX:

```
MessageWaitTime
```

If the number of milliseconds specified by **MessageWaitTime** is exceeded, the client will generate an **swTimeoutErr** error.

The minimum value you can set MessageWaitTime is 500 (milliseconds) — smaller values will automatically be changed to 500. An exception to the minimum value is the special value of 0 (zero) — if MessageWaitTime is set to 0, the client will not timeout.

Be aware that the user under which the client is running *must* have read access to the MessageWaitTime Registry value, otherwise it will silently ignore the setting (if you are running under IIS, by default, the user does not have access). Use the **regedit** utility to grant access to the Registry value.

MessageWaitTime is a global setting — all programs will use this single setting. There is no means to set a message wait time for individual programs.

Also note that if you view message wait times in the client log, they are shown in the log as “Message Segment Wait Time(*value*)”.

Character Encoding Using ICU Conversion Libraries

ICU conversion libraries can be used to specify the desired character encoding.

To use the ICU conversion libraries, you must create the following environment variable (UNIX systems) or Registry entry (Windows systems) and set it to the name of the converter you wish to use.

— TISOUnicodeConverterName

On Windows systems, the Registry entry must be located in the following path:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Staffware plc\Staffware SSO Client\
```

Note - If you install 32-bit software on a 64-bit machine, the Registry path will include "Wow6432Node" as follows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Staffware plc\...
```

For a list of converter names, and information about each converter, see the following website:

http://demo.icu-project.org/icu-bin/convexp

Note that when using the ICU libraries, the converter you use must reserve positions 00 through 1F for the standard single-byte ASCII control characters. This ensures that the control characters do not otherwise occur in the byte stream. (The UTF-16 converter, for example, does not satisfy this requirement, and therefore, cannot be used.)

UNIX Systems

If the **TISOUnicodeConverterName** environment variable does not exist, or is set to an invalid value, the ICU libraries are not used. In this case, the system looks for the **TISOMultiChar** environment variable. If the TISOMultiChar environment variable exists and is set to 1, UTF-8 (multi-byte) encoding is used, otherwise extended ASCII (single-byte) encoding is used. The system will only look at the TISOMultiChar environment variable if the TISOUnicodeConverterName environment variable does not exist or is set to an invalid converter name.

Windows Systems

If the **TISOUnicodeConverterName** Registry entry does not exist, or is set to an invalid value, the ICU libraries are not used. In this case, the system looks for the **TISOMultiChar** Registry entry in the following path:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Staffware plc\Staffware SSO Client\
```

Note - If you install 32-bit software on a 64-bit machine, the Registry path will include "Wow6432Node" as follows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Staffware plc\...
```

If the TISOMultiChar Registry entry exists and is set to 1, UTF-8 (multi-byte) encoding is used, otherwise extended ASCII (single-byte) encoding is used. The system will only look at the TISOMultiChar Registry entry if the TISOUnicodeConverterName Registry entry does not exist or is set to an invalid converter name.

For more information about ICU, see: http://icu.sourceforge.net/

19

XML Interface

Introduction

The TIBCO iProcess Server Objects (Java) provides an XML interface to allow you to retrieve all data from the TIBCO iProcess Engine as an XML data stream.

The XML interface is provided in the following package:

- com.staffware.sso.xml

The XML Server Objects are thin wrappers around the "non-XML" Server Objects. Each time you make a method call on an XML Server Object, the **xSession** object will receive the results, and add them to an array of **vResult** objects, one for each method call. Upon request, the array is passed to the standard serialization object, getting back an XML data stream. The output is routed to the data stream object that was passed to the **xSession.getXMLResults** method as a parameter. By using the data stream, the caller has complete control over the destination of the XML data.

This is described in more detail in the following subsections.

XML Server Objects

The **com.staffware.sso.xml** package contains the objects that implement the TIBCO iProcess Server Objects functionality. These "XML Server Objects" are as follows:

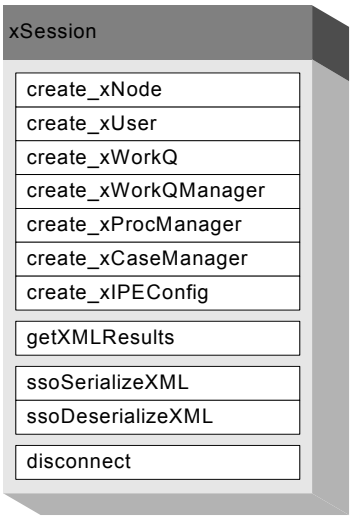
- **xSession** - This object must be instantiated first — all of the other XML Server Objects are then created from it (with the exception of xNodeManager, which is not created via the xSession object). For more information, see *Using the XML Interface* on page 295.
- **xCaseManager** - Used to manage cases.
- **xNode** - Provides access to information stored on the node (users, groups, roles, etc.).
- **xNodeManager** - Used for node discovery and log settings.
- **xProcManager** - Provides access to procedure definitions.
- **xUser** - Provides access to a user's work queue and work items.
- **xWorkQ** - Used to access and process work items in a work queue.
- **xWorkQManager** - Used to manage work queues.
- **xIPEConfig** - Provides access to database and activity monitoring configurations.

These XML Server Objects contain most of the same methods and return the same data as their non-XML-equivalent Server Objects (although in XML format rather than object format). The exception is that the XML Server Objects do not contain any methods that return pageable lists; new "make" and "fetch" methods have been added to support list access. For more information, see *Returning Lists of Items* on page 299.

Using the XML Interface

To retrieve XML data from the TIBCO iProcess Engine, you must first instantiate an **xSession** object. Once you have an xSession object, you can then create the other XML Server Objects using the "**create_**<*xmlServerObject*>" methods on xSession. The other Server Objects cannot be instantiated directly in the XML interface.

You must pass a "node context" object (**vNodeCtx**) as a parameter in the xSession object constructor. The vNodeCtx object identifies the TIBCO iProcess Objects Server you will be connecting to, as well as the user name and password you will be connecting as. Therefore, each xSession object represents a single connection to the server.



Constructing the xSession Object

Two constructors are provided for the xSession object, allowing you to use the one that is appropriate for your situation:

Constructor 1:

```
xSession(SWServerImplType aServerImpl,
          vNodeCtx aNodeCtx)
```

where:

- *aServerImpl* specifies whether JBase or RMI objects will be used. The **SWServerImplType** enumeration is used to specify either **swJBase** or **swRMI**. This allows you to move from in-process objects (JBase) to external objects (RMI) without requiring you to recompile your code.

  Important - If RMI is specified in this parameter, the JNDI parameters must already be set up in a **JNDI.properties** file, or specified using **-D** parameters when the Java Virtual Machine is started (see *Registry Location* on page 22 for more information). Also, if RMI is used, the naming registry and Server Factory must be running prior to constructing xSession.

- *aNodeCtx* identifies the TIBCO iProcess Objects Server to connect to, as well as the user name and password used to connect to the server. You must construct this object prior to constructing the xSession object.

Constructor 2:

```
xSession(vNodeCtx, aNodeCtx,
          String aSF_Name,
          boolean aSF_IsJRMP,
          Hashtable aSF_JNDIEnv)
```

where:

- *aNodeCtx* identifies the TIBCO iProcess Objects Server to connect to, as well as the user name and password used to connect to the server. You must construct this object prior to constructing the xSession object.
- *aSF_Name* specifies a Server Factory name. Passing an empty string for the Server Factory name (*aSF_Name*=“”) causes the default (**ssoServerFactory**) to be used.
- *aSF_IsJRMP* specifies the export protocol, either JRMP or IIOP (if *aSF_IsJRMP*=True, use JRMP; if *aSF_IsJRMP*=False, use IIOP).
- *aSF_JNDIEnv* is a hashtable that contains explicit JNDI environment settings to use when creating the context used to locate the Server Factory. Passing a NULL for the JNDI environment means to use the default JNDI context settings.

Note that the second constructor implies RMI.

XML Results

When using the XML interface, individual method calls do not directly return XML data, nor do they throw errors. Instead, the results from method calls (including exceptions) are added to an array of **vResult** objects internal to the **xSession** object. Notice that all method calls on the XML Server Objects return void and expect a “results ID” (*aId* parameter) as an input parameter. For example:

```
void getStartProcIds(String aId)
```

The results ID is a locator reference for the caller. Every call to an XML Server Object method takes a results ID as a parameter; that ID is reflected back in the **Id** attribute of the **<vResult>** tag in the XML data (see the example in the *Example XML Data* on page 297). (Note that if a NULL is passed in the *aId* parameter, the results ID defaults to the name of the method call with an initial capital letter — in this example, it would be “GetStartProcIds”.)

To get the XML data stream, you must call the **getXMLResults** method on xSession, passing in a stream object to which you would like the XML data stream written:

```
void getXMLResults(OutputStream aStream,
                   boolean aWithInput,
                   boolean aClear)
```

where:

- *aStream* specifies the data stream object to which you would like the results routed. This can be a memory stream, a file stream, or whatever type of stream is appropriate.
- *aWithInput* specifies whether or not to include the **<InParam>** tag, and its corresponding input data, in the XML data stream. Pass True in this parameter to include the input parameters in the XML data; pass False to exclude it.
- *aClear* specifies whether or not to clear the array of **vResult** objects on the server after returning the results.

*Note - The **getXMLResults** method is also available on the **xNodeManager** object so you can get results from calls to methods on xNodeManager (the xNodeManager object is not created via xSession).*

Example XML Data

The XML data is returned to the output stream with a single root tag: **<vSSOData>**.

Every **<vSSOData>** tag has as its first level of children:

- a **<NodeId>** tag
- a **<LoggedInUserName>** tag
- a **<Results>** tag

The **<Results>** tag contains a **<vResult>** tag for each method called on the XML Server Object.

The **<vResult>** tag has an **Id** attribute that contains the results ID passed in on the method call. This is a locator reference for the caller. Every call to an XML Server Object takes a results ID as a parameter; that ID is reflected back in the **<vResult>** tags' **Id** attribute.

The **<vResult>** tag may also have an **<InParam>** tag that contains the input parameters to the method call. If the method took no parameters, there will be no **<InParam>** tag. Also note that the **<InParam>** tag is included in the XML data only if the *aWithInput* parameter is True when the **getXMLResults** method is called.

Following is an example of XML data that would be returned by the XML interface:

```
<?xml version="1.0"?>
<sso:vSSOData xmlns:sso="http://tibco.com/bpm/sso/types">
  <sso:NodeId>
    <sso:Name>i2tagtest</sso:Name>
    <sso:ComputerName>ozquadling</sso:ComputerName>
    <sso:IPAddress>10.97.8.55</sso:IPAddress>
    <sso:TCPPort>58002</sso:TCPPort>
    <sso:IsDirector>false</sso:IsDirector>
  </sso:NodeId>
  <sso:LoggedInUserName>swadmin</sso:LoggedInUserName>
  <sso:Results>
    <sso:vResult Id="MyRequestId">
      <sso:AWorkQs>
        <sso:vAWorkQ>
          <sso:Name>user0002</sso:Name>
          <sso:Description>User 0002</sso:Description>
          <sso:HostingNode>i2tagtest</sso:HostingNode>
          <sso:Tag>i2tagtest|user0002|R</sso:Tag>
          <sso:IsGroup>false</sso:IsGroup>
          <sso:IsReleased>true</sso:IsReleased>
          <sso:FirstDeadline>3000-12-31T23:15:00.0000000</sso:FirstDeadline>
          <sso:DeadlineCnt>0</sso:DeadlineCnt>
          <sso:UnopenedCnt>9</sso:UnopenedCnt>
          <sso:UrgentCnt>0</sso:UrgentCnt>
          <sso:WorkItemCnt>11</sso:WorkItemCnt>
          <sso:WorkQParam1Name>WQ Parameter1</sso:WorkQParam1Name>
          <sso:WorkQParam2Name>WQ Parameter2</sso:WorkQParam2Name>
          <sso:WorkQParam3Name>WQ Parameter3</sso:WorkQParam3Name>
          <sso:WorkQParam4Name>WQ Parameter4</sso:WorkQParam4Name>
          <sso:Redirection>
            <sso:StartingDateTime>
              <sso:IsValueSet>false</sso:IsValueSet>
              <sso:DateTime>0001-01-01T00:00:00.0000000</sso:DateTime>
```

```
            </sso:StartingDateTime>
            <sso:EndingDateTime>
              <sso:IsValueSet>false</sso:IsValueSet>
              <sso:DateTime>0001-01-01T00:00:00.0000000</sso:DateTime>
            </sso:EndingDateTime>
            <sso:WorkQName />
          </sso:Redirection>
        </sso:vAWorkQ>
        <sso:vAWorkQ>
          <sso:Name>user0001</sso:Name>
          <sso:Description>Staffware user</sso:Description>
          <sso:HostingNode>i2tagtest</sso:HostingNode>
          <sso:Tag>i2tagtest|user0001|R</sso:Tag>
          <sso:IsGroup>false</sso:IsGroup>
          <sso:IsReleased>true</sso:IsReleased>
          <sso:FirstDeadline>3000-12-31T23:15:00.0000000</sso:FirstDeadline>
          <sso:DeadlineCnt>0</sso:DeadlineCnt>
          <sso:UnopenedCnt>0</sso:UnopenedCnt>
          <sso:UrgentCnt>0</sso:UrgentCnt>
          <sso:WorkItemCnt>0</sso:WorkItemCnt>
          <sso:WorkQParam1Name>WQ Parameter1</sso:WorkQParam1Name>
          <sso:WorkQParam2Name>WQ Parameter2</sso:WorkQParam2Name>
          <sso:WorkQParam3Name>WQ Parameter3</sso:WorkQParam3Name>
          <sso:WorkQParam4Name>WQ Parameter4</sso:WorkQParam4Name>
          <sso:Redirection>
            <sso:StartingDateTime>
              <sso:IsValueSet>false</sso:IsValueSet>
              <sso:DateTime>0001-01-01T00:00:00.0000000</sso:DateTime>
            </sso:StartingDateTime>
            <sso:EndingDateTime>
              <sso:IsValueSet>false</sso:IsValueSet>
              <sso:DateTime>0001-01-01T00:00:00.0000000</sso:DateTime>
            </sso:EndingDateTime>
            <sso:WorkQName>Tellers</sso:WorkQName>
          </sso:Redirection>
        </sso:vAWorkQ>
      </sso:AWorkQs>
      <sso:InParam>
        <sso:AWorkQContent>
          <sso:IsWithParticipation>false</sso:IsWithParticipation>
          <sso:IsWithRedirection>true</sso:IsWithRedirection>
          <sso:IsWithSupervisorNames>false</sso:IsWithSupervisorNames>
          <sso:IsWithCDQPDefs>false</sso:IsWithCDQPDefs>
        </sso:AWorkQContent>
        <sso:WorkQTags>
          <sso:string>i2tagtest|user0002|R</sso:string>
          <sso:string>i2tagtest|user0001|R</sso:string>
        </sso:WorkQTags>
      </sso:InParam>
    </sso:vResult>
  </sso:Results>
</sso:vSSOData>
```

Returning Lists of Items

When using the XML interface, you can access lists (i.e., blocks) of items on the server using the **make**<*type*>**List** and **fetch**<*type*>**List** methods. ("Pageable lists" — an older method of accessing lists of items on the server — are not available in the XML interface.)

For information about the using **make**<*type*>**List** and **fetch**<*type*>**List** methods, see *Using Single-Block Item Access* on page 137.

When you call the **make**<*type*>**List** and **fetch**<*type*>**List** methods, the requested block of items is returned internally to the **xSession** object — you must call the **getXMLResults** method to send the block of items to the XML data stream. For more information, see *XML Results* on page 296.

Dates and Times

Methods that return dates and times (e.g., vACase.getTimeStarted, vDateTime.getDateTime, etc.) return them in the following serialized format in the XML interface:

```
"yyyy'-'mm'-'dd'T'hh':'mm':'ss'.0000000'"
```

For example:

```
2005-06-29T10:30:28.0000000
```

Notes about dates:

- The milliseconds are always 0000000.
- There is no time zone indicator.
- For values objects, the seconds are not preserved.
- The date and time will be what is stored on the server — no adjustments for the caller locale are made.

Error Handling

The methods on the XML server classes do not throw exceptions. The results of all XML method calls, whether successful or not, are returned as part of the XML stream. All functions are defined as returning void, with the exception of the **make**<xxx>**List** methods, which return the HeldId as a string (if there was an error creating the list, the error will be in the XML and the HeldId returned will be an empty string).

The reason that all results, successful or not, are returned in the XML stream is because:

- There might be multiple method calls before the serialized response is requested. Some of these calls might be successful and some might not. Trying to handle exceptions outside the XML stream would be complicated.
- The code making the XML request may not be the code that needs the result. Therefore, it is not in the proper program context to deal with errors.

If the XML method call encounters an exception in the call to the function, that exception will be returned in the **<Exception>** element within the **<vResult>** for that call. (Calls can be matched to their results using the Result's “Id” attribute.)

Below is an example of the **getFieldDefs** call returning a “Procedure name not found” exception.

```
<?xml version="1.0" encoding="UTF-8"?>
<sso:vSSOData xmlns:sso="http://tibco.com/bpm/sso/types">
    <sso:NodeId>
        <sso:Name>i2tagtest</sso:Name>
. . .
    </sso:NodeId>
    <sso:LoggedInUserName>swadmin</sso:LoggedInUserName>
    <sso:Results>
        <sso:vResult Id="getFieldDefs">
⇒           <sso:Exception>
                <sso:ExceptionDetails/>
                <sso:ErrorGroup>swSEOServerException</sso:ErrorGroup>
                <sso:ErrorCode>swNoProcErr</sso:ErrorCode>
                <sso:Message>Procedure name not found.  </sso:Message>
                <sso:StackTrace></sso:StackTrace>
            </sso:Exception>
        </sso:vResult>
    </sso:Results>
</sso:vSSOData>
```

In more complex methods where an array of parameter values are passed, each of which is acted on individually, the result is more complicated:

- If there were no errors, the results are serialized as regular data and no **<Exception>** element is returned.
- If any parameter generates an error, a **vEx**<objectClass> exception is thrown.

See the examples below.

In the following example, we see the work queues, “user0001” and “user0002” being successfully requested. The results are returned within the **<AWorkQs>** element.

```
<?xml version="1.0"?>
<sso:vSSOData xmlns:sso="http://tibco.com/bpm/sso/types">
  <sso:NodeId>
    <sso:Name>i2tagtest</sso:Name>
. . .
  </sso:NodeId>
  <sso:LoggedInUserName>swadmin</sso:LoggedInUserName>
  <sso:Results>
    <sso:vResult Id="MyRequestId">
      <sso:AWorkQs>
⇒       <sso:vAWorkQ>
          <sso:Name>user0002</sso:Name>
. . .
        </sso:vAWorkQ>
⇒       <sso:vAWorkQ>
          <sso:Name>user0001</sso:Name>
. . .
        </sso:vAWorkQ>
      </sso:AWorkQs>
    </sso:vResult>
  </sso:Results>
</sso:vSSOData>
```

However, if the work queues could not be accessed, an **<Exception>** tag is returned. Each parameter that generates an error will be returned within its own **<vExceptionDetail>** element. Below we see that the first parameter in the array "**<ArrayIndex>0**" was an invalid work queue name ("yada_yada_yada") and that the second parameter in the array had a length of 0 (an empty string).

Since there was no valid data, the **<AWorkQs>** element is empty.

```
<?xml version="1.0"?>
<sso:vSSOData xmlns:sso="http://tibco.com/bpm/sso/types">
  <sso:NodeId>
    <sso:Name>i2tagtest</sso:Name>
. . .
  </sso:NodeId>
  <sso:LoggedInUserName>swadmin</sso:LoggedInUserName>
  <sso:Results>
    <sso:vResult Id="MyRequestId">
      <sso:AWorkQs />
      <sso:Exception>
        <sso:ExceptionDetails>
⇒         <sso:vExceptionDetail>
            <sso:ArrayIndex>0</sso:ArrayIndex>
            <sso:ErrorCode>swInvalidParamErr</sso:ErrorCode>
            <sso:ErrorGroup>swParameterException</sso:ErrorGroup>
            <sso:ParameterInfo>yada_yada_yada</sso:ParameterInfo>
            <sso:Message>Invalid Parameter. Invalid WorkQ Name in Tag</sso:Message>
          </sso:vExceptionDetail>
⇒         <sso:vExceptionDetail>
            <sso:ArrayIndex>1</sso:ArrayIndex>
            <sso:ErrorCode>swInvalidParamErr</sso:ErrorCode>
            <sso:ErrorGroup>swParameterException</sso:ErrorGroup>
            <sso:ParameterInfo />
            <sso:Message>Invalid Parameter. WorkQ Tag length is 0</sso:Message>
```

```
          </sso:vExceptionDetail>
        </sso:ExceptionDetails>
        <sso:ErrorGroup>swParameterException</sso:ErrorGroup>
        <sso:ErrorCode>swItemErrErr</sso:ErrorCode>
        <sso:Message>One of the items in the array returned an error.  </sso:Message>
        <sso:StackTrace></sso:StackTrace>
      </sso:Exception>
    </sso:vResult>
  </sso:Results>
</sso:vSSOData>
```

In the situation where the parameters of the call generate some good results and some errors, you get an XML result like the example below.

The first work queue tag has the unknown work queue name “yada_yada_yada” and generates an **<Exception>** element with a single **<vExceptionDetail>** below it. The second work queue tag “user0001” returns the valid work queue data found beneath the **<vAWorkQ>** element. The **<ArrayIndex>** identifies the element in the input parameter array that caused the error.

```
<?xml version="1.0"?>
<sso:vSSOData xmlns:sso="http://tibco.com/bpm/sso/types">
  <sso:NodeId>
    <sso:Name>i2tagtest</sso:Name>
. . .
  </sso:NodeId>
  <sso:LoggedInUserName>swadmin</sso:LoggedInUserName>
  <sso:Results>
    <sso:vResult Id="MyRequestId">
      <sso:AWorkQs>
⇒       <sso:vAWorkQ>
          <sso:Name>user0001</sso:Name>
. . .
        </sso:vAWorkQ>
      </sso:AWorkQs>
      <sso:Exception>
        <sso:ExceptionDetails>
⇒         <sso:vExceptionDetail>
            <sso:ArrayIndex>0</sso:ArrayIndex>
            <sso:ErrorCode>swInvalidParamErr</sso:ErrorCode>
            <sso:ErrorGroup>swParameterException</sso:ErrorGroup>
            <sso:ParameterInfo>yada_yada_yada</sso:ParameterInfo>
            <sso:Message>Invalid Parameter. Invalid WorkQ Name in Tag</sso:Message>
          </sso:vExceptionDetail>
        </sso:ExceptionDetails>
        <sso:ErrorGroup>swParameterException</sso:ErrorGroup>
        <sso:ErrorCode>swItemErrErr</sso:ErrorCode>
        <sso:Message>One of the items in the array returned an error.  </sso:Message>
        <sso:StackTrace></sso:StackTrace>
      </sso:Exception>
    </sso:vResult>
  </sso:Results>
</sso:vSSOData>
```

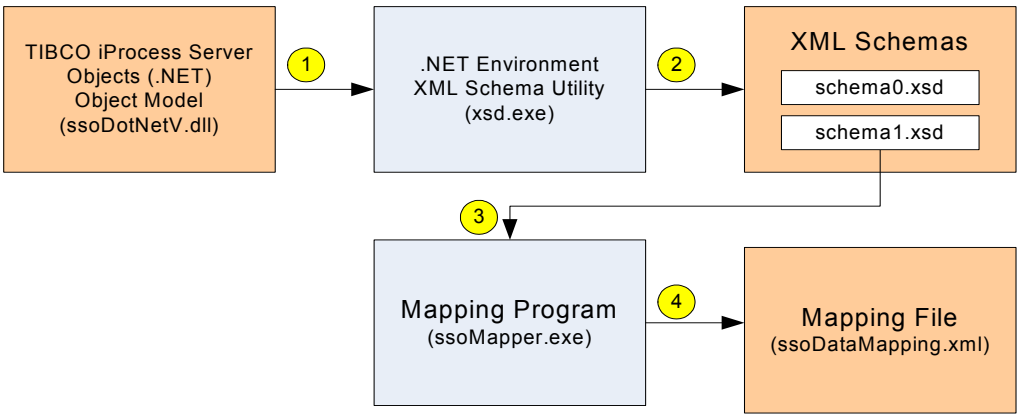
Object Serialization

The standard java serialization objects do not produce the desired XML structure. Therefore, we are using the serialization objects from Castor (see http://www.castor.org), which we have customized, as well as an object mapping file, to serialize and deserialize Value Objects to XML.

The desire is to use the same XML schema for both of the TIBCO iProcess Server Objects products: Java and .NET. But, unfortunately, Castor's open-source serialization objects do not produce XML that is identical to that of .NET. Therefore, we are using Castor's mapping facility to map the TIBCO iProcess Server Objects (Java) object model to the desired XML — so that it matches .NET.

The following steps describe how the mapping file is produced:

1. The TIBCO iProcess Server Objects (.NET) object model is used as input to the Microsoft XML schema utility (**xsd.exe**), which is provided in the .NET development environment.
2. The **xsd.exe** utility produces two XML schema files (**schema0.xsd** and **schema1.xsd**). Only the **schema1.xsd** file is used — this file is located in the same directory in which the mapping file is located; see step 4.
3. The **schema1.xsd** file is used as input to a utility that TIBCO developed to generate a Castor mapping file from the schema.
4. The mapping program produces a mapping file (**ssoDataMapping.xml**) that maps the data in the TIBCO iProcess Server Objects (Java) objects to the correct XML based on the XML schema.



The **schema1.xsd** schema file and the **ssoDataMapping.xml** mapping file are distributed with the XML serialization classes — they are located in the same location in which the **xNodeManager.class** is loaded, but under an additional "resources" subdirectory. For example:

```
...classes\com\staffware\sso\xml\xNodeManager.class
...classes\com\staffware\sso\xml\resources\ssoDataMapping.xml
```

These files are distributed in the **ssoXMLSerializer.jar** file in the TIBCO iProcess Server Objects (Java) installation directory.

*Note - As the **schema1.xsd** file describes the structure of the TIBCO iProcess Server Objects (Java) XML interface, it can be used by someone who wants to program directly to the XML interface (it is also used internally by the Action Processor if you are using the TIBCO iProcess Workspace (Browser) product).*

Serialize/Deserialize Functions

The normal way to use the TIBCO iProcess Server Objects XML interface is to allow the **xSession** object to accumulate Value Objects returned from calls to Server Object method calls. However, you may have a need to serialize and deserialize Value Objects for some other reason. To accommodate this need, the following Static utility functions are available on the **xSession** object:

- **ssoSerializeXML** - This takes in a Value Object and writes the XML serialized object to the passed in data stream.
- **ssoDeserializeXML** - This takes in a data stream that contains iProcess Server Objects-compatible XML, and returns it deserialized into a Value Object.

These functions are on **xSession** because it has already loaded the mapping file in Java, and can manage the deserialization.

If you are using these functions to serialize/deserialize, note that Castor cannot deserialize XML that has an array as its root. To work around this problem, use the **vInParam** object, which contains a variable for every type that can be passed as a parameter to a method call. Simply set the appropriate variable on **vInParam** and serialize it. When deserializing, ensure that the root element in the XML is **<InParam>** and deserialize it to a **vInParam** object.

Index

Symbols
‘?’ character 78
‘*’ character 78, 242, 268
’^’ character 241, 267
’?’ (question mark), in REs 215, 243
’.’ (period), in REs 242, 268
’*’ (asterisk), in REs 215, 243
’$’ character 242, 268
%DESC 71, 80
%USER 71, 80
$SWDIR environment variable 5
$undeliv work queue 126

A
Abandoned period parameter 158
Abort, transaction control step 98
aCaseFieldNames parameter 163, 199, 227, 253
Access
  another user’s work queue 129
  authority 70, 187
  field definitions 165
  permissions 156
aCDQPNames parameter 163
Activating/deactivating
  client log 288
Active cases 68
Activity monitoring 42
addCaseAuditEntry method 80
Adding
  entries to the client log 291
  user-defined audit trail entries 80
  users to group 185
  work queue supervisors 134
Addressee of work item 208, 234, 260
addUsersToGroups method 185
ADMIN access 187
Admin name 41
AF_ALL_SUBSTART 80
AF_OUTSTANDING_ONLY 80
aFieldsOption parameter 122, 163
aHoldList parameter 155
aIsReturnAllCDQPs parameter 110, 163
aIsReturnAllFields parameter 109, 163
aIsWithAuditData parameter 72
AND operator 195, 223, 249
Application Servers 23
Arguments, in method calls 13
Array fields 172
Arrays 14
Arrival date and time 208, 234, 260, 275
ASCENDING ARRIVAL sort order 188
ASCENDING DEADLINE sort order 188
Asterisk, in REs 242, 268
Attachments 171
"Attribute not found" error 189
Attributes
  creating definition 189
  deleting 189
  modifying value 189
  user 187
Audit
  data
    getting when filtering cases 206, 232, 258
  data filtering 75
  procedure versions 51
  step objects 70
  sub-procedures 74
  trail 69
Audit filter expression 75
AUDIT_TYPE filter criteria 75
audit.mes file 69
Auditing cases 69
auditusr.mes file 69, 80
Authentication, WebShhere JAAS 24
Authority
  system administrator 187, 190
  user, general 190
Available count 156, 196, 224, 250
aWIFieldNames parameter 121, 163

B
Bean implementation 30
Bind name 17, 20
Block size, pageable lists 159
Buffers
  message 154

C
Cancelling
  a redirection schedule 133
Case 2

administration authority 190
auditing 69
closed, make active 102
closing 102
data 125
controlling 163
filtering on 212, 238, 264
setting 167
vs. work item data 166
data fields 109
sorting on 277
description 62, 208, 234, 260, 275
family 100
management 61
number, obtaining 65
prediction 85
purging 102
reference number 208, 234, 260, 275
start authority 66
starting 61
suspending 100
Case data
getting, impact of 199, 227, 253
Case Data Queue Parameter fields
See CDQP fields
CaseFieldNames parameter 109, 163, 199, 227, 253
Cases
active 68
closed 68
determining number of 68
getting list of 67
retrieving 153
sorting 273
Castor 303
CDQP fields
defined 169
filtering on 109, 212, 238, 264
sorting on 277
specifying in work item content 109, 163
CDQPNames parameter 110, 163
changeAttributes method 189
changeDefaultCriteria method 219, 245, 271, 279
Changing
user's password 183
Character encoding 179, 293
CheckOSUser configuration parameter 183
Classpath 22
clear method 159
clearDefaultCriteria method 219, 245, 271, 279
Clearing blocks 159
Client log 283
activating/deactivating 288
adding entries to 291
filtering 288
location 287
resetting 291
setting size 291
closeCases method 102
closeCasesByCriteria method 102, 218, 245, 270
Closed cases 68
Closing cases 102
based on filter criteria 218, 245, 270
com.staffware.sso.data 17, 19
com.staffware.sso.ejb 29
com.staffware.sso.jbase 17, 19
com.staffware.sso.rmi 19
com.staffware.sso.util 17, 19
Command-line options 20
Commit and Concede 98
Commit and Continue 98
Comparison operators 200, 204, 227, 230, 253, 257
Completing a graft step 96
Computer name 33
Configuring
TCP port 38
Connection ID, database 41
Connection, TCP 11
Constructing
Delegate Objects 23, 31
Server Objects 11
Value Objects 15
vDate and vTime objects 131
vDateTime objects 134
vNodeId objects 33
vSortField objects 274
xSession object 295
Content filtering
when retrieving work items 109
when retrieving work queues 107
Content Request Objects 8, 161
Context factory 22
Controlling
Case Data 163
fields returned when locking work items 163
Conversions, data type when filtering 211, 237, 263
CORBA 17

Counts, for pageable lists 156, 196, 224, 250
createAttributeDefs method 189
createGroups method 184
createRoles method 186
createUser method 182
Creating
  attribute definition 189
  iProcess user 182
  participation schedules 130
  role 186
  Server Objects 12
  user group 184
Criteria
  objects 9, 194, 222, 248

D

"-D" option 23
Data type
  conversions, filtering 211, 237, 263
  sorting by 280
  used in filtering 211, 236, 263
  validation 170
Database case filtering 193, 221, 247
Database configuration 41
Date
  data type 211, 236, 263
  format 178
    in XML interface 299
  in filter expressions 217, 244, 269
Date released, procedure version 50
DATE_RANGE filter criteria 79
DateTime
  data type 211, 236, 263
  in filter expressions 217, 244, 270
DBConnectionAccess 41
Deadline 127
  count 157, 196, 224, 251
  date and time 208, 234, 260, 275
  expired flag 208, 234, 260, 275
  recalculating 128
  set flag 209, 234, 260, 275
  time 275
Default
  filter criteria 219, 245, 270
  JNDI context 23
  sort criteria 279
Defining filter expressions 195, 223, 249
Delegate Objects 8, 13, 23, 30
deleteAttributeDefs method 189
deleteGraftTask method 95
deleteGroups method 185
deleteRoles method 186
deleteUserPreference method 192
deleteUsers method 183
Deleting
  iProcess user 183
  role 186
  user attribute 189
  user group 185
  work item on withdrawal 129
Delta
  count 111
  status 112
Dependent objects, retrieving 162
Deploying SSO beans 29
Deployment descriptor file 29
DESCENDING ARRIVAL sort order 188
DESCENDING DEADLINE sort order 188
DESCRIPTION attribute 187
Deserialize function 304
Directed UDP Message 34
Director 5
Directory, client log file 287
Directory, system 5
DISABLE_USER_CHECK process attribute 183
disableCategory method 290
disableMessage method 291
disconnect method 11, 157
Duration property 86
Dynamic
  deadlines, recalculating 128
  sub-procedure call step, outstanding 82
  sub-procedures 53
  TCP port 38

E

EAI step, outstanding 82
EAI steps 98
EJBHome remote interface
  Home interface 30
EJBObject remote interface 30
Empty
  dates and times 131, 134
  fields, in filtering 216, 244, 269
enableCategory method 289
enableMessage method 291
Encoding, character 179, 293
Enterprise bean implementation 30
Enterprise JavaBeans (EJB) configuration 3, 16, 27

Enumeration Type Objects 9
Environment variables
  client log 286
Error
  handing 281
    in XML interface 300
  objects 8
  processing
    dynamic sub-procedures 54
    graft steps 96
  result of processing work item 127
Escape characters, in filter expressions 216, 243, 269
Event publishing 42
Event step, outstanding 82
Event, triggering 85
Exception element, XML 300
Exceptions 281
Exclude count 156, 196, 224, 251
Executing command
  on work item keep 125
  on work item release 126
Expired deadline 127
Export protocol 20, 23, 31
Expressions property 66, 70
External
  processes 92
  work items 135
External processes 95
External Rebind 20
externalGraftProcessComp method 92, 95

F

Factory, naming 22
fetchAWorkItemListDelta method 111
fetchAWorkItemListJMS method 116, 118
fetchList method 137
fetchWorkItemListDelta method 111
Field
  data, retrieving from the server 167
  iProcess 165
  uninitialized 167
  validations 170
Fields
  passing to/from sub-procedure 56
FieldsOption parameter 122, 163
FILTER_FLAGS filter criteria 80
Filtering 194, 222, 248
  audit steps 75
  cases in database 193, 221, 247
  client log 288
  content
    when accessing work items 109
    when retrieving work queues 107
  data types 211, 236, 263
  expression format 206, 232, 258
  expression length 250
  for dependent objects 160
  on case data 212, 238, 264
  on deadline information 128
  on empty fields 216, 244, 269
  on ranges of values 217, 244, 269
  predicted items 90
  using system fields 208, 234, 260
  work items 109
  work items in WIS 193, 221, 247
Firewall, using with SSO 40
Form markings 170
Format, of date 178
Forwardable work item flag 209, 234, 261, 275
Forwarding
  permission 132
  work queues 131
forwardItems method 131

G

Garbage collection 11, 157
getACaseList method 67, 195, 223, 249, 274
getACaseListHeld method 67, 155
getACases method 67
getAction method 71
getActiveCnt method 68
getActivityPub method 43
getAdminByUserRef method 70
getAFMarkings method 170
getAGroupList method 184
getAGroupListHeld method 155, 184
getAGroups method 184
getAttributeDefs method 188
getAttributes method 188
getAuditProcIds method 67
getAuditProcs method 70
getAuditSteps method 72
getAvailableCnt method 156, 196, 224, 250
getAWorkItemList method 108, 148, 195, 223, 249, 274
getAWorkItemListHeld method 108, 155
getAWorkItemListJMS method 108, 113, 148
getAWorkItemListJMSHeld method 108, 155
getAWorkItems method 108

getAWorkQList method 106
getAWorkQListHeld method 106, 155
getAWorkQs method 106
getCaseCnt method 68
getCaseDescOpt method 62
getCaseFields method 165, 166
getCaseNumber method 65
getCDQPDefs method 277
getCDQPDefs methods 212, 238, 264
getCDQPs method 212, 238, 264, 277
getClientLog method 285
getClosedCnt method 68
getCPQPs method 90
getCustomAuditMsgDefs method 81
getDatabaseConfig method 41
getDateCreated method 50
getDateModified method 50
getDateReleased method 50
getDateWithdrawn method 50
getDeadline method 127
getDeadlineCnt method 157, 196, 224, 251
getDefaultCriteria method 219, 245, 271, 279
getDeltaCnt method 111
getDeltaStatus method 112
getDurationValues 86
getEndingDateTime method 133
getErrorCode method 282
getExceptionDetails method 282
getExcludeCnt method 156, 196, 224, 251
getExternalForm method 135
getExternalGraftProcesses method 94
getExternalId method 135
getFieldDefs method 165
getFieldsOption method 122, 164
getFieldType method 170
getFirstDeadline method 127
getFormMarkings method 170
getForwardToWorkQIds 132
getFRows method 170
getGroupAttributes method 188
getGroups method 184
getHeldId method 155
getInFromFldNames method 56
getInitialExpr method 124
getInstance method 35, 36
getInToFldNames method 56
getInvalidCnt method 156, 196, 224, 251
getItem method 150
getItemsPerBlock method 156, 159, 196, 224, 251
getKeepExpr method 125
getLastUpdateUser method 50
getLocalCnt method 156, 196, 224, 251
getLockedBy method 124
getMajorVersion method 49
getMenu method 187, 189
getMenuName method 187, 189
getMessage method 71
getMinorVersion method 49
getMsgNumber method 81
getMsgTemplate method 81
getNextDeltaItem method 111
getNodeId method 152
getNodes method 35
getOSUserList method 149
getOSUserListHeld method 155
getOutFromFldNames method 56
getOutstandingItems method 83, 94, 129
getOutToFldNames method 56
getOverMaxCnt method 156, 196, 224, 251
getParticipations method 130
getPredictedCondition method 87
getPredictedItemList method 87, 195, 223, 249, 274
getPredictedItemListHeld method 155
getProcAudits method 51
getProcDefs method 47
getProcId method 67
getProcIds method 67
getProcMajorVersion method 49
getProcMinorVersion method 49
getProcPath method 83
getProcVersions method 50
getPublicDescription method 60
getPublicFields method 59
getPublicSteps method 59
getRedirection method 133
getReleaseExpr method 126
getRetryDateTime method 99
getRetryDelay method 99
getReturnStatus method 95
getReturnStatusFld method 54
getRoleNames method 66, 70
getRoles method 186
getSessionId method 11
getStartByUserRef method 66
getStartIndex method 58
getStartingDateTime method 133
getStartProcIds method 67
getStartProcs method 66

getStartStepName method 61
getStatus method 49, 66
getSubCaseId method 74
getSubProcCases method 94
getSubProcName method 52
getSubProcNameFld method 53, 92
getSubProcPath method 57
getSubProcStartStep method 52
getSubProcStartStepFld method 53
getSuccessDetails method 282
getSupervisedQIds method 106, 134
getSupervisorNames method 134, 187
getTag method 67
getTaskCnt method 93
getTransactionType method 99
getType method 63, 86, 170
getUDPPortNumbers method 34, 36
getUnopenedCnt method 124, 157, 196, 224, 251
getUrgentCnt 157, 196, 224, 251
getUsageURL method 60
getUserAttributes method 188
getUserList method 149, 182
getUserListHeld method 155, 182
getUserNames method 66, 70
getUserPreference method 192
getUsers method 182
getVersionComment method 50
getWorkItemFields method 121, 165
getWorkItemList method 108, 148, 195, 223, 249, 274
getWorkItemListHeld method 108, 155
getWorkItems method 108
getWorkItemTag method 121
getWorkQ method 106
getWorkQIdList method 106, 148
getWorkQIdListHeld method 106, 155
getWorkQIds method 106
getWorkQList method 106, 148
getWorkQListHeld method 106, 155
getWorkQName method 133
getWorkQParam#Name methods 278
getWorkQs method 106
getXMLResults method 294
Graft steps 91
    outstanding 82
Groups 2, 184
    retrieving 153

H
Held
    ID 155
    pageable lists 155, 159
Hidden case description 62
hold method 155
HoldList parameter 155
Hood ID 284
Hosting node 2

I
IAP JMS library 42
IAPConfigAccess 42
ICU conversion libraries 179, 293
IDX_ 173
IIOP export protocol 20, 31
ImplicitMoveSysInfo parameter 181
IMPMONITOR command 42
Incomplete status 48
Indexed collection 150, 153
Indexes, array fields 173
Inheritance 14
Initial factory 22
InParam tag 296
Instantiating
    Server Objects 11
    Value Objects 15
Interfaces 16
Internet Inter-Orb Protocol (IIOP) 17
Invalid count 156, 196, 224, 251
IP address 33
IPEADMIN user 41, 182, 190
iProcess
    users 182
iProcess Engine 4
iProcess field 165
isActive method 288
isArrayField method 172
IsAuditAscending parameter 73
isDeadline method 127
isDeadlineAWD method 127
isDeadlineExp method 127
isHaltOnSubProc method 54, 96
isHaltOnTemplate method 54, 96
isHaltOnTemplateVer method 54, 96
isIgnoreCaseSuspend flag 101
isKeepLocalItems flag 150, 154, 156, 196, 224, 251
isKeepOnWithdrawal method 129
isLocked method 122

isLongLock method 122
isLongLocked method 123
isMandatory method 60
isOrphaned method 126
isOutstanding method 71, 94
isPrediction method 88
isPublic method 59
isReleasable method 126
isReleased method 105
IsReturnAllCDQPs parameter 110, 163
IsReturnAllFields parameter 109, 163
isSuspended flag 101
isTaskCntSet method 93
isUndelivered method 126
isUnopen method 124
"isWith" parameters 162
IsWithAuditData parameter 72
isWithForwardToWorkQIds 132
isWorkQReleased method 105
ItemsPerBlock parameter 154, 159

J
JAAS authentication 24
JAR file
    RMI 19
    standalone 17
Java Message Service (JMS) 42
Java Naming and Directory Interface (JNDI) 17
Java Remote Method Protocol (JRMP) 17
Java Virtual Machine 22, 23, 24, 157
Java.Exception 281
java.io.Serializable 14
java.naming.factory.initial 22
java.naming.provider.url 22
javax.ejb.SessionBean 30
JBase 3, 16
jbase package 17, 19
JMS topic, work queue deltas 112
JNDI 17
    context, default 23
    environment settings 23, 296
JNDI.Properties 22, 23
JNDI.properties 295
JRMP export protocol 20, 31
jumpTo method 81
JumpToStepName parameter 74
JVM 22, 23, 24, 157

K
keepAliveWorkQDeltaJMSPublications method 114, 116, 118
Keeping
    start step 62
    work item 2, 124
        and passing field data 169
keepItems method 169

L
LANGUAGE attribute 187
Length, of filter expressions 250
List state objects 144
Lists
    single-block item access 137
Lists, pageable 148
Local count 156, 196, 224, 251
Lock vs. long lock 122
Locker of the work item 209, 234, 261, 275
lockFirstAWorkItem method 123
lockFirstWorkItem method 123
Locking work items 121
    with field data 168
lockItems method 121, 163
Log level
    client log 288
log method 291
Logging
    client 283
    client messages 283
Logical operators 200, 204, 227, 230, 253, 257
LogID, client log 287
LogLevel
    Client log 288
Long lock 122

M
Major version number 48, 49
makeAWorkItemListJMS method 115, 117
makeList method 137
makeTag method 50, 67
MANAGER access 187
Managing
    cases 61
    work queues 104
Mapping file, serialization 303
Markings 170
    types 63
    validating 63
Maximum count, over 156, 196, 224, 251
Memo fields 171
MENUNAME attribute 187, 189
Message buffers 154
Message Event Request (MER) 42

Message, audit trail 71
Messages
filtering in client log 290
MessageWaitTime 292
Minor version number 48, 49
Model status 48
Modifying
a redirection schedule 133
participation schedules 130
user attribute value 189
Monitoring activities 42
MOVESYSINFO function 181
moveSysInfo method 181
Multi-byte encoding 293

N
Naming
conventions 6, 294
factory 22
New work item flag 210, 235, 262, 276
Node
management 32
name of SPO Server 33
NoPersistedIdErr 157
Normal step, outstanding 82
NULL 162
Number of
Cases 68
Numeric data type 211, 236, 263

O
O/S user 180
Object
serialization, XML 303
timeout monitor 21
Types 8
objectMonitor 11, 21
ODBC connection name 41
Operators
comparison 200, 204, 227, 230, 253, 257
logical 200, 204, 227, 230, 253, 257
Optional case description 62
OR operator 195, 223, 249
Oracle SID 41
Orphaned work item 126
OSUsers
retrieving 153
Outstanding
graft items 94
items
determining 82
withdrawing 81
step 71
Over Maximum Count 156, 196, 224, 251

P
Pack data 124, 166
Packages
EJB 29
RMI 19
standalone 17
Pageable list 148
controlling resources 157
counts 156, 196, 224, 250
held 155
refreshing 151
Parallel steps 166
Parameter templates 54, 96
Parameters, used in method calls 13
Participating in another user's work queue 129
Participation schedule 130
Password
changing 183
database user 41
Permission
to access held pageable lists 156
to audit 69
Persisting
filter criteria 219, 246, 271
sort criteria 279
PLHeldTimeOut 158
PollCnt parameter 35, 37
Precedence, starting sub-procedures 64
predictCase method 88
Predicting cases 85
Prediction flag 88
Priority of work item 209, 235, 261, 276
Procedure 1, 46
audit trail 51
determining who can audit 69
name 209, 235, 261, 276
number of cases of 68
status 48
sub-cases of 52
version control 48
which can user start 66
PROCEDURE sort order 188
Process
Engine 4
Invocator service 182

ProcPath 83
PRODEF access 187
Protocol, export 20, 23, 31
Provider URL 22
Provider, database 41
Public steps 59
Public steps, fields 59
Publishing events 42
purgeAndReset method 102
purgeByCriteria method 218, 245, 270
purgeCases method 102
purgeCasesByCriteria method 102
Purging cases 102
    based on filter criteria 218, 245, 270

Q
QSUPERVISOR attribute 187
QSUPERVISORS attribute 134
Queue not found error 105

R
Range filtering 217, 244, 269
rebind method 24
Rebind, external 20
Rebuilding a pageable list 151
Recalculating deadlines 128
Redirecting work items 133
Redirection schedule 133
Reentrant 15
refresh method 35, 110, 151
Refreshing a pageable list 151
Registry
    client log 285
    location 22
Regular expressions 215, 241, 267
Releasable
    work item flag 126, 209, 235, 261, 276
Released status 48
Released work queue 104
releaseEAIItem method 135
releaseItems method 125, 169
releaseResources method 157
Releasing
    start step 62
    the start step 126
    work item 2, 125
Remote
    interface 19
        implementation 19
    Method Invocation (RMI) configuration 3, 16, 17
RemoteException 30
remove method 11
removeSupervisors method 135
removeUsersFromGroups method 185
Removing
    participation schedules 130
    users from group 185
    work queue supervisors 135
Required case description 62
resetLog method 291
Resetting
    case counter 102
    the client log 291
Resources, controlling 11, 157
ResumedDescription parameter 74
ResumedStepName parameter 74
Retrieving
    cases, work queues, users from the server 153
    dependent objects 162
    field data from the server 167
    work items from server 108, 149
    work queues 106
Return status
    sub-procedures 54, 59
Return status, graft step 95
RMI 3, 16
RMI configuration 17
Roles 186
root user
    access to system directory 5
Router 37
rsServerFactory 17
rsServerFactoryImpl 11
rsServerFactoryImpl.class 20

S
sBase object 11
sCaseManager 10
Schedule
    participation 130
    redirection 133
Schema utility 303
schema1.xsd 303
sClientLog 285
Script, running when
    keeping work item 125
    locking work item 124
    releasing work item 126
Sending

directed UDP messages 34
UDP broadcast 35
Serialization 14
Serialization, XML 303
Server
Factory 17, 20
implementation 20
Objects 8, 10
services file 39
SessionBean 30
Sessions 11
setActive method 288
setActivityPub method 43
setCaseData method 167
SetCaseSuspended method 100
setCategories method 289
setGraftTaskCnt method 93
setIsKeepLocalItems method 159
setLogDirectory method 287
setLogId method 287
setLogLevel method 288
setMaxSize method 291
setMessages method 290
setUDPPortNumbers method 34, 36
setUserPreference method 192
SF_IsJRMP parameter 23, 31
SF_JNDIEnv parameter 31
SF_Name parameter 23, 31, 296
Sharing between threads 15
Sharing user sessions 12
SI Interface 16
simulateCase method 88
Single-block item access lists 137
Single-byte encoding 293
sIPEConfig 10, 41
sIPEConfig object 42
Size of
blocks, pageable lists 159
client log 291
sNode 10
sNodeManager 10, 34
constructing 35
Sort criteria 201, 206, 228, 231, 254, 258
Sorting
on case data 276
on Case Data fields 277
predicted items 90
work items 109
work items and cases 273
SORTMAIL attribute 188
SortTypeAs parameter 280
sPageableList 148
sPageableListJ object 113
sPageableListR 148
Specifying filter criteria 194, 222, 248
sProcManager 10, 46, 59
SQL select statement 193, 221, 247
sSession 10, 12
ssoAllMarkings 122, 164
SSOClient log file 287
ssoDataMapping.xml 303
ssoDeserializeXML 304
ssoFieldList 122, 164
ssoFormMarkings 122, 163
ssoJBase.jar 17, 19
ssoRMI.jar 19
ssoSerializeXML 304
ssoServerFactory 17, 23, 296
ssoXMLSerializer.jar file 303
StackSize parameter 250
staffo file/database table 182, 184
staffpms file 178, 183
Start step
for sub-procedures 53
keeping/releasing 62, 126
startCase method 61, 166
StartCaseDescription parameter 73
StartCaseStepName parameter 73
Started date and time of the case 209, 235, 261, 276
Starter of the case 209, 235, 261, 276
startGraftTask method 92
StartIndex 58, 84
Starting
a case 61
who has access 66
with field data 166
a graft task 92
startWorkQDeltaJMSPublish method 114, 116, 117
Static TCP port 38
Status
delta 112
of pageable list 151
of the case 209, 235, 261, 276
sub-procedures 59
Step 2
duration 86
Forward 132
parallel 166

public 59
STEP_DESC filter criteria 79
STEP_NAME filter criteria 79
Studio 1, 46
Sub-Cases 52
Sub-procedure 52
auditing 74
dynamic 53
outstanding 82
precedence 64
start steps 53
Supervisors
of work queues 134
sUser 10
SuspendedDescription parameter 74
SuspendedStepName parameter 74
Suspending cases 100
SW_CASENUM 65
SW_DEADLINE 128
SW_DEADLINEDATE 128
SW_DEADLINETIME 128
SW_EXPIRED 128
SW_GEN_IDX 173
SW_HASDEADLINE 128
SW_NA 167, 216, 244, 269
sw_NoPersistedIdErr 157
SW_QPARAM1-4 213, 239, 265
SW_QPARAMn fields 278
SW_STARTER 126
swAdded 112
swAttachment field types 171
SWAuditActionType 43, 71, 74, 101
swCaseOnly 128
SWConditionPredictType 87
swDateSort 280
swDateTimeSort 280
SWDeadlineCalculateType 128
swDeleted 112
swDeltaTooBigErr 110
SWDescOptionType 62
SWDIR, system directory 5
swDurationDeadline 87
SWDurationType 86
swDynamicReleased 74
swDynamicWithdrawn 74
SWEmptyField 165, 167, 175
swErrBadSubProc 74
swErrDiffTemplate 74
swErrDiffTemplateVer 74
SWFieldsOptionType 122
SWFieldType 170
swHiddenDesc 62
swIncludeSubCases 128
swIncomplete 48, 66
SWLogCategoryType 289
SWLogMessageType 290
SWMarkingType 63
swMemo field types 171
swModel 48
swModified 112
SWMonitorList.xsd schema 42, 43
swNoReCalc 128
swNotDeltaItem 112
swNumericSort 280
swOptionalDesc 62
sWorkQ 10
sWorkQManager 10
SWPageableListType 144
swPLChanged 151
swPLNoChange 151
swPLOrphaned 151
SWPLRefreshType 151
swPLStatusOnly 151
SWPLStatusType 151
swPrecedenceR 64
SWProcAuditActionType 51
swProcessedTo 71
SWProcStatusType 49
swRecreate 151
swReleased 48, 66
swRequiredDesc 62
SWServerImplType enumeration 295
SWSortType 280
swStartCase 71
swSubCaseComp 74
swSubCaseExpired 74
swSubCaseStart 74
swSubCaseTerm 74
swSubCaseWithdrawn 74
SWSubProcPrecedenceType 64
SWSubProcStatusType 54, 95
swTextSort 280
swTimeSort 280
SWTransactionType 99
swUnreleased 48, 66
swUpdate 151
swUpdateWithDelta 110, 151
swuser user 182
swutil function 277
swWithdrawn 48, 66

swWithdrawnIncomplete 48
System
  administrator 187, 190
  directory 5
  fields 171
    list of for sorting 275
    used in filtering 208, 234, 260
    using in sorting 275
System-defined audit trail messages 69

T
Tag, getting/making 67
TCP
  connection 11
  port 33
    configuring 38
    for database 41
Templates, audit trail messages 81
Terminated date and time of the case 210, 235, 261, 276
TerminationDescription parameter 73
TerminationStepName parameter 73
TerminationUser parameter 73
Test work queue 104
Text data type 211, 236, 263
Third-party application 135
Threads, sharing objects 15
TIBCO
  iProcess Engine 4
  iProcess Objects Director 5
  iProcess Objects Server 5
    service 37
  iProcess Studio 46
  Process Engine 4
TIBCO Business Studio 1
Time
  data type 211, 236, 263
  format in XML 299
  in filter expressions 217, 244, 269
Timeout monitor 21
Timeout, server reply 292
TISOMultiChar 293
TISOUnicodeConverterName 179, 293
TNS connection 41
Transaction control steps 98
  outstanding 82
Transparent Network Substrate (TNS) 41
triggerEvent method 85
  recalculating deadlines 128
  resurrect case with 102
Troubleshooting 281
"type mismatch error" 170
Type validation 170
Types of users 180

U
UDP
  broadcast 35
  message 34
  port 34, 36
UDP_SERVICE_NAME process attribute 34, 36, 37
UDPServiceName Parameter 34, 36, 37
Undelivered work item 126
Undoing work item changes 124
undoItems method 124
Uninitialized field 167
Unlocking a work item 123
unlockItems method 123
Unopened count 157, 196, 224, 251
Unopened work item flag 210, 235, 262, 276
Unreleased status 48
Urgent count 157, 196, 224, 251
Urgent work item flag 210, 235, 262, 276
User 2
  access authority 187
  administration 180
  attributes 187
    creating 189
    deleting 189
    modifying value 189
  authority 190
  creating 182
  Datagram Protocol (UDP) 34, 35
  deleting 183
  groups 184
  Message, in client log 284
  preference data 192
  session 11
  types 180
USER access 187
USER_NAME filter criteria 78
User-defined audit trail messages 69
USERFLAGS attribute 188
USERINFO 181
Users
  add/remove from group 185
  iProcess 182
  retrieving 153
UTF-8 293

V
vACaseContent 72, 161, 163, 167
vACaseCriteria 9, 194, 222, 248, 273
vAccessUserRef 66, 70
vAFMarking 170
vAGroup 184
vAGroupContent 161, 184
Validating
    fields/markings 63, 125, 170
Value Objects 8, 14
    constructing 15
vAProcContent 161
vAttribute 188
vAttributeDef 188
vAuditMsgDef 81
vAuditStep 70
vAWIContent 132, 161
vAWorkItem 108
vAWorkQ 107
vAWorkQContent 107, 161
vCDQP 212, 238, 264, 277
vCDQPDef 107, 212, 238, 264, 277
vClientLog 285
vDatabaseConfig 41
vDate 131
vDateTime 134
vDeadline 127
vDuration 86
vDurationValue 86
vDynamicSubProcItem 82
vEAIItem 82, 135
Verbose, command-line option 21
verifyNode method 34, 37
Version control, procedures 48
vEventItem 82
vException 282
vExternalGraftProcess 94, 95
vField 165
vFieldDef 165
vFMarking 170
vFRow 170
vGraftItem 82, 94
vGraftStep 92
vGraftSubTask 92
vGroup 184
vGroupContent 161
vGroupId 184
vListState 144
vListState objects 144
vNode 35
vNodeCtx 295
vNodeCtx object 32
vNodeId 33
vNormalItem 82
vOutstandingItem 83
vOutstandingItemContent 83, 161
vParticipation 107, 130
vPredictedItem 85
vPredictionCriteria 90, 194, 222, 248, 273
vPreference object 192
vProcAudit 51
vProcDef 47
vProcDefContent 47, 51, 161
vProcSummary 68
vPublicField 60
vPublicStep 59
vRedirection 107, 133
vResult tag 296
vRole 186
vSortField 274
vSSOData tag 297
vStepContent 60, 161
vStepId 59
vSubProcItem 82, 94
vSummary 146
vTime 131
vTransactionControlItem 82, 99
vTransactionControlStep 98
vUser 182
vUserContent 161
vWIContent 109, 162, 163, 167, 199, 227, 253
vWICriteria 9, 194, 219, 222, 245, 248, 270, 273, 279
vWIFGContent 121, 162, 163, 167
vWIFieldGroup 121, 163, 168
vWILocked object 123
vWISummary 146
vWorkItem 108
vWorkItemId 108
vWorkQ 107
vWorkQId 107, 161

W
WebSphere JAAS authentication 24
WIContent 109
WIFieldNames parameter 121, 163
Wild card characters
    filtering cases 205
    filtering work items 200
Wildcard characters

Filtering audit data 78
filtering cases 231, 257
filtering work items 227, 254
WIS work item filtering 193, 221, 247
WIS_QCHANGE_MAX_CHANGES 110, 113
WIS_QCHANGE_MAX_PCT 110, 113
WIS-compatible 208, 275
Withdrawing
outstanding items 81
work item 127, 129
Withdrawn status 48
WithdrawnIncomplete status 48
Work item 2
changes 110
data 124, 213, 239, 265
vs. case data 166
deadlines 127
deltas 110
determining if new (unopened) 124
determining who locked 124
errors 127
external 135
filtering and sorting 109
forwarding 131
keeping 124
locking 121
orphaned 126
processing 121
releasable 126
releasing 125
retrieving 108, 149
sorting 273
undelivered 126
undoing changes 124
unlocking 123
withdrawing 129
Work queue 2
deltas 110
via JMS topic 112
forwarding 131
manager
filter criteria 219, 246, 271
managing 104
parameter fields
for filtering 213, 239, 265
parameter fields, sorting on 278
participating 129
redirecting 133
retrieving 106, 153
supervisor 134
test/released 104
"Work Queue not found" error message 183
WorkQContent 107
WQSAbandonedPeriod parameter 158

X

xCaseManager 294
xIPEConfig 294
XML
interface 294
results 296
serialization 303
xNode 294
xNodeManager 294
xProcManager 294
xsd.exe 303
xSession object 294
xUser 294
xWorkQ 294
xWorkQManager 294