

TIBCO iProcess™ Objects Programmer's Guide

Version 10.4

May 2010

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 LICENSE.PDF) 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.

TIB, TIBCO, TIBCO Adapter, Predictive Business, Information Bus, The Power of Now, TIBCO ActiveMatrix BusinessWorks, and TIBCO iProcess are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

EJB, Java EE, J2EE, JMS and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. 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.TXT 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 © 2000-2010 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Table of Contents

Preface	xiv
Introduction	xiv
Product Name Changes	xiv
Knowledge Level	xv
Documentation Set	xv
Revision History	xvi
Chapter 1 — Introduction	1
Introduction	1
Procedures	1
TIBCO iProcess Objects Configurations	3
TIBCO iProcess Objects Components	4
TIBCO Process / iProcess Engine	5
Engine and Server Version Numbers	5
SWDIR - The System Directory	6
Engine Processes	6
TIBCO iProcess Objects	7
Client-to-Server Communications	7
TIBCO iProcess Objects Server	7
TIBCO iProcess Objects Director	8
JVM Finalization and Garbage Collection	8
Solution Strategy	9
Implementation	9
Configuration	10
FinalizeMonitor Log Example	13
Chapter 2 — The Object Model	14
Introduction	14
Chapter 3 — Naming Conventions	17
Overview	17
Differences between COM, Java, and C++ Naming Conventions	17
Chapter 4 — Getting Started	19
Creating the SWEnterprise Object	19
Do I Create One or Many SWEnterprise Objects?	19
When Should the SWEnterprise Object be Destructed?	19
Accessing Nodes on the Network	20
Automatic Discovery (UDP Broadcast) on a LAN Segment	22
Configuring the UDP Broadcast	23
Multiple Instances of the TIBCO iProcess Objects Server/Director	23
What if a Known Node is not Answering the UDP Broadcast?	24
Example — Auto-Discovery UDP Broadcast	24
Directed UDP to a Specific Node	25
Specifying the UDP Port Number	25

Example — Directed UDP Broadcast	26
Manually Create an SWNodeInfo Object	26
Example — Connecting to a Specific Node	27
Configuring the TIBCO iProcess Objects Server TCP Port	27
Configuring the TCP Port on a Windows System	28
Configuring the TCP Port on a UNIX System	29
Can I use TIBCO iProcess Objects through a Firewall?	30
Creating Enterprise Users	31
Logging In	32
Turning On/Off Password Checking	33
Logging In When Using Multiple Instances of the TIBCO iProcess Objects Server	33
Login Failures	33
Logging in Using a TIBCO iProcess Objects Director	34
Logging Out	34
How often should Users be Logged In and Out?	34
When Should I use Anonymous Logins?	35
Database Configuration	36
Database Configuration Access	36
Activity Publication	37
Activity Publication Access	37
Configuring Activity Publication	37
Using the SWIPEConfig Object	38
Configuration Example	38
Chapter 5 — Procedures	41
Introduction	41
Procedure Version Control	42
Accessing the Procedure Version Number	42
Procedure Status	43
Listing Versions of a Procedure	44
Accessing a Specific Procedure Version	44
Making Different Versions of Procedures	44
Using the Tag Property to Make Specific Versions	45
Procedure Version Details	45
Procedure Audit Trails	46
Sub-Procedures	46
Sub-Procedure Call Steps	47
Sub-Procedure Start Precedence	47
Dynamic Sub-Procedure Call Steps	48
Passing Data between a Main and Sub-Procedure	50
Outstanding Sub-Procedures / Sub-Procedure Call Steps	50
Sub-Procedure Proc Path	51
Public Steps	53
Chapter 6 — Working with Lists	55
Types of Lists Available	55
How to Determine the Type of Object in a List	56
Lists are Filled Asynchronously	57
How to Force Synchronous Behavior	57
Determining the Number of Objects in a List or View	58

What about SWXLists?	58
SWLists	59
How SWLists are Created and Populated at the Client	59
What Causes Raw Data to be Sent to the Client?	60
What Causes Objects to be Created and Added to an SWList?	61
Why do Item and ItemByKey return a Variant (COM only)?	61
When Should I Rebuild an SWList?	62
How to Rebuild Subordinate Lists	62
SWLocLists	63
How to Add Objects/Strings to Local Lists	63
How to Access Objects/Strings on Local Lists	64
Why do Item and ItemByKey return a Variant (COM only)?	64
SWViews	65
The “Default” Views of Cases and Work Items	65
How SWViews are Created and Populated at the Client	66
What Causes Raw Data to be Sent to the Client?	66
What Causes Objects to be Created and Added to an SWView?	68
When should a View be Rebuilt?	69
How to Rebuild Subordinate Views	70
When should I Create an Alternate View?	70
Implied Sort on Alternate Views	71
Determining the Total Number of Items Available	71
How do I Limit the Number of Work Items/Cases in a View?	72
How to Include Audit Data in a View	72
SWXLists	73
What types of Objects can be found in an XList?	73
Accessing the “Default” XLists of Objects	74
Creating Additional XLists	75
How XLists are Created	75
Controlling Resources	77
Populating an XList of Work Items	77
SWXLists of Work Items When Using Multiple Instances of the Server	79
Populating an XList of Cases	79
Populating an XList of Groups, Users, or OSUsers	80
Determining the Number of Items in an XList	80
Work Item-Specific Counts	81
Why do Item and ItemByKey return a Variant (COM only)?	81
Working with Persisted XLists	81
Using Multiple Instances of the Server or Director	82
How to Include Audit Data in an XList	83
CIList (Java Only)	83
Using an SWCIList	83
Object Keys	84
Chapter 7 — Fields & Markings	86
What is a Staffware Field?	86
What are Markings?	88
Type Validation on Fields/Markings	89
User-Created Markings	90
Type Validation on User-Created Markings	90
Case Data vs. Work Item Data	91

Where do I find Case Data and Work Item Data?	91
Setting Case Data	92
Keeping/Releasing the Start Step	92
Parallel Steps	92
Case Data Queue Parameter Fields	93
Accessing Memo Fields	93
Accessing Attachments	93
Accessing System Fields	94
Array Fields	94
Array Field Indexes	95
Using Array Fields in Filter Expressions	96
Date Format	97
Chapter 8 — Filtering Work Items and Cases (without Filtering Enhancements)	98
Introduction	99
How Filtering Differs Between Views and XLists	100
Defining Filter Expressions on SWView	100
Number of Work Items or Cases in the Filtered View	100
Defining Filter Expressions on SWXList	101
Number of Work Items or Cases in the Filtered XList	102
Filtering/Sorting in an Efficient Manner	103
Filtering/Sorting Work Items	103
Getting Case Data	105
Can the WIS Perform the Filter Operation?	105
Can the WIS Perform the Sort Operation?	107
Filtering/Sorting Cases	107
Getting Case Data	109
Filtering Cases on the TIBCO iProcess Objects Server	109
Sorting Cases on the TIBCO iProcess Objects Server	110
Getting Audit Data	111
Filter Criteria Format	111
System Fields used in Filtering	113
Data Types Used in Filter Criteria	116
Data Type Conversions	116
Filtering on Case Data Fields	117
Using Case Data Queue Parameter Fields	117
Type of Data in CDQPs	118
Using Work Queue Parameter Fields	118
Work Queue Parameter Fields vs. Case Data Queue Parameter Fields	119
Using Regular Expressions	120
Using Escape Characters in the Filter Expression	121
Filtering on Empty Fields	121
How to Specify Ranges of Values	122
Specifying Multiple Ranges	122
Closing/Purging Cases Based on Filter Criteria	123
How to Persist (Default) Filter Criteria	124
Chapter 9 — Filtering Work Items and Cases (with WIS WorkItem Filtering)	126
Introduction	127
How Filtering Differs Between Views and XLists	128

Defining Filter Expressions on SWView	128
Number of Work Items or Cases in the Filtered View	128
Defining Filter Expressions on SWXList	129
Number or Work Items or Cases in the Filtered XList	130
Filtering/Sorting in an Efficient Manner	131
Filtering/Sorting Work Items	131
Getting Case Data	133
Work Items are Filtered by the WIS	133
Can the WIS Perform the Sort Operation?	134
Filtering/Sorting Cases	134
Getting Case Data	136
The TIBCO iProcess Objects Server Filters Cases	136
The TIBCO iProcess Objects Server Sorts Cases	137
Getting Audit Data	138
Filter Criteria Format	138
System Fields used in Filtering	140
Data Types used in Filter Criteria	142
Data Type Conversions	143
Filtering Work Items on the WIS	143
Filtering Cases on the TIBCO iProcess Objects Server	143
Filtering on Case Data Fields	144
Using Case Data Queue Parameter Fields	144
CDQPs Contain Work Item Data	145
Using Work Queue Parameter Fields	145
Work Queue Parameter Fields vs. Case Data Queue Parameter Fields	146
Using Regular Expressions	147
Regular Expressions with Work Item Filtering	147
Using Escape Characters in the Filter Expression	148
Filtering on Empty Fields	149
How to Specify Ranges of Values	149
Closing/Purging Cases Based on Filter Criteria	150
How to Persist (Default) Filter Criteria	150
Chapter 10 — Filtering Work Items and Cases (with WIS WorkItem & Database Case Filtering). 152	
Introduction	153
How Filtering Differs Between Views and XLists	154
Defining Filter Expressions on SWView	154
Number of Work Items or Cases in the Filtered View	154
Defining Filter Expressions on SWXList	155
Number or Work Items or Cases in the Filtered XList	156
Length of Filter Expressions	156
Large Filter Expressions May Require Larger Stack Size in UNIX	157
Filtering/Sorting in an Efficient Manner	157
Filtering/Sorting Work Items	157
Getting Case Data	159
Work Items are Filtered by the WIS	159
Can the WIS Perform the Sort Operation?	160
Filtering/Sorting Cases	160
Getting Case Data	162
The Database Filters Cases	162
The Database Sorts Cases	163

Getting Audit Data	163
Filter Criteria Format	163
System Fields used in Filtering	165
Data Types used in Filter Criteria	168
Data Type Conversions	168
Filtering on Case Data Fields	169
Using Case Data Queue Parameter Fields	169
CDQPs Contain Work Item Data	170
Using Work Queue Parameter Fields	170
Work Queue Parameter Fields vs. Case Data Queue Parameter Fields	171
Using Regular Expressions	172
Regular Expressions with Work Item Filtering	172
Regular Expressions with Case Filtering	173
Using Escape Characters in the Filter Expression	174
Filtering on Empty Fields	174
How to Specify Ranges of Values	175
Closing/Purging Cases Based on Filter Criteria	176
How to Persist (Default) Filter Criteria	176
Chapter 11 — Sorting Work Items and Cases	178
Introduction	178
How Sorting Differs Between Views and XLists	178
Defining Sort Criteria on SWView	178
Defining Sort Criteria on SWXList	179
Specifying Sort Criteria	180
Sorting in an Efficient Manner	182
System Fields used in Sorting	182
Sorting on Case Data Fields	184
Using Case Data Queue Parameter Fields	184
CDQPs Contain Work Item Data	185
Using Work Queue Parameter Fields	185
Setting Default Sort Criteria	186
Implied Sort Fields for Multiple Views/XLists	188
Sorting as a Specified Data Type	188
Chapter 12 — Managing Work Queues	189
Introduction	189
Work Queue Objects	189
Test vs. Released Work Queues	190
Accessing Work Items on a Work Queue	190
Accessing Work Items in SWViews	191
Accessing Work Items in SWXLists	192
Determining the Number of Work Items in a Work Queue	192
Determining the Number of Work Items in a Work Queue on an SWView	192
Determining the Number of Work Items in a Work Queue on an XList	193
Processing Work Items	194
Locking Work Items	194
Getting Markings When Locking Work Items	194
What’s the Difference Between a “Lock” and a “Long Lock”?	195
Unlocking a Work Item	195
Discarding Changes made to a Locked Work Item	195

Has a Work Item been Locked/Opened?	196
Determining who Locked a Work Item	196
Executing a Command when a Work Item is Locked	196
Keeping Work Items	196
Executing a Command when a Work Item is Kept	197
Releasing Work Items	197
Validating Markings	197
Executing a Command when a Work Item is Released	197
Automatically Releasing the Start Step	198
What is an Orphaned Work Item?	198
Determining if a Work Item could not be Delivered to the Addressee	198
Is the Work Item Directly Releasable?	198
Errors Resulting from Processing Work Items	198
Forwarding Work Items to Another Work Queue	199
Manually Forwarding Work Items	199
Auto Forwarding/Redirecting Work Items	200
Redirection of Work Items	200
Creating a Redirection Schedule	201
Using the SWDate Object (Java and C++ Clients Only)	202
Granting Access to a Work Queue	203
View-Only Access to a Work Queue	203
Participation Access to a Work Queue	204
Creating a Participation Schedule	205
Using the SWDate Object (Java and C++ Clients Only)	205
The TIBCO iProcess Objects Server Maintains an Index of the Participation Schedules	207
Work Queue Supervisors	208
Adding Work Queue Supervisors	208
Removing Work Queue Supervisors	208
Work Item Deadlines	209
Withdrawing Work Item on Deadline	209
Deadline Counts	210
Filtering and Sorting on Deadline Information	210
Keeping a Work Item that is Withdrawn	211
External Work Items	212
Releasing an External Work Item	212
Chapter 13 — User Administration	214
Introduction	214
Types of Users	215
MOVESYSINFO Function	215
Staffware Users	216
Creating a Staffware User	216
Deleting a Staffware User	217
Is an O/S User needed for every Staffware User?	217
Changing the User's Password	217
User Groups	218
Creating a User Group	218
Deleting a User Group	219
Adding and Removing Users to/from a Group	219
Roles	220

Creating a Role	220
Deleting a Role	220
User Attributes	221
Creating an Attribute	223
Deleting an Attribute	224
Modifying an Attribute	224
Why isn't the new User, Group, Role or Attribute Appearing in the List?	224
Determining which Procedures a User can Audit	225
Determining the Procedures for which the User can Start a Case	226
User Authority	227
Chapter 14 — Case Management	230
Starting a Case	230
Case Description	230
Keeping/Releasing the Start Step	231
Starting a Case with Field Data	232
Validating Markings on the Start Step	232
Sub-Procedure Precedence	233
Why isn't the Started Case Appearing in the Work Queue?	233
Obtaining the Case Number of a Case that was just Started	233
Determining Who Can Start a Case	234
Which Procedures can a User Start?	235
Obtaining the "Default" View / XList of Cases	236
Status of Cases	237
Creating an "Alternate" View / XList of Cases	237
Alternate SWViews	237
Implied Sort on Alternate Views	237
Alternate SWXLists	238
Status of Cases	238
Determining the Number of Cases in a Procedure	238
Auditing Case Data	239
Determining the Procedures a User can Audit	240
Populating a Case with Audit Data	241
The SWAuditStep Object	242
Configuring Audit Trail Strings	242
Auditing Sub-Procedures	243
Filtering Audit Data	245
Setting AuditFilterExpr	245
Adding User-defined Audit Trail Entries	246
Predicting Cases	247
Defining Case Prediction	248
Step Duration	248
Conditional Actions for Case Predictions	249
Performing Case Prediction	249
Background Case Prediction	249
Live Case Prediction	249
Case Simulation	250
Sub-Procedures, Dynamic Sub-Procedures, and Graft Steps in Prediction	250
Sub-Procedure Call Steps	250
Dynamic Sub-Procedure Call Steps and Graft Steps	250
Including Case Data Queue Parameter Data in Prediction Results	251

Filtering and Sorting Predicted Items	252
Triggering Events	253
Suspending Cases	254
Reactivating a Suspended Case	254
Ignoring Suspended Cases	255
Jumping To New Outstanding Step in a Case.	255
Determining Outstanding Items	256
ProcPath to Outstanding Items	257
Using Graft Steps	259
Defining Graft Steps.	259
Starting a Graft Task.	260
Setting the Task Count	261
Outstanding Graft Items	261
Return Statuses	262
Other Status Information on an Outstanding Graft Item	263
Deleting a Task	263
Completing a Graft Step	263
Error Processing	263
Transaction Control Steps.	265
Step Type	265
Type of Transaction Control Step.	266
Outstanding Transaction Control Steps	267
Retrying Failed Transactions	267
Transaction Control Step Audit Trail Messages.	268
Closing Cases	268
Resurrecting a Closed Case	268
Purging Cases	269
Chapter 15 — Stateless Programming	270
Introduction.	270
Logging Out vs. Disconnecting	270
Stateless Objects	271
Single-Parameter Methods	272
Using the Tag Property	272
Passing an Empty Object in the Make Methods (C++ only)	273
Persisted XLists.	274
Chapter 16 — Optimizing Your Applications	275
Introduction.	275
Handling Large Lists of Work Items, Cases, Users, OS Users, Groups.	275
Clear Blocks on Client when using XLists.	276
Optimizing Communications between Client and Server.	276
Filtering and Sorting in an Efficient Manner	277
How Getting Case Data affects Application Efficiency	277
Getting Case Data on View/XList vs. Case	277
Looping Through Items in an SWList or SWView	278
Locking, Keeping, Releasing Multiple Items	278
Optimizing VB Application Performance.	279
Accessing a Single Object.	279
Clear Unneeded Views and XLists	279
Case Indexing	279

Chapter 17 — Client Configuration	280
Client Log	280
Controlling the Client Log	282
The SWLog Object	282
Registry Settings (Windows only)	283
Environment Variables (UNIX Only)	283
Name and Location of the Client Log	284
Log File Name	284
Log File Directory	284
Activating / Deactivating the Client Log	284
Filtering the Client Log	285
Setting the Log Level	285
Filtering by Category	285
Filtering by Message	287
Displaying Memory Information in the Client Log (Windows only)	288
Adding Entries to the Client Log	288
Setting the Size of the Client Log	288
Resetting the Client Log	288
Testing the Client Log	289
Message Wait Time	290
Character Encoding Using ICU Conversion Libraries	291
Chapter 18 — Error Handling	292
TIBCO iProcess Objects (COM)	292
Error Constants	292
Error Trapping in Visual Basic	293
TIBCO iProcess Objects (Java)	294
Error Constants	295
TIBCO iProcess Objects (C++)	296
Handling Multiple Work Item Operation Errors	296
Debugging Problems with ASP	297
Error Messages	298
“An Exception of Type java.lang.UnsatisfiedLinkError was not handled”	298
“Authentication Request Failed”	298
“Error 2140: An internal Windows NT error occurred”	298
“Error 2140: An internal Windows NT error occurred”	299
“Error 2140: An internal Windows NT error occurred”	299
“Error 2140: An internal Windows NT error occurred”	301
“Error 2140: An internal Windows NT error occurred”	302
“Error 5: Access is denied”	302
“Error Calling CreateDataSource Interface for SWAutoFwdQ”	302
“Error Initializing AutoFwd/QView Database”	302
“Error creating mutex”	303
“Error in sal_frm_putdata call”	304
“One of the items in the array returned an error”	304
“The memory could not be ‘written’”	304
“Unable to locate DLL”	304
“/usr/lib/dld.sl exists - can't open shared library: /oracle8/lib/libclntsh.sl no such file or directory”	305
“Work Item is not accessible”	305

Chapter Appendix A — Code Examples	306
Introduction	306
Auto-Discovery UDP Broadcast	307
Visual Basic	307
Using the “For Each” Iteration:	307
Using a While loop:	307
Java	308
Using “Enumeration”:	308
Using a While Loop:	308
C++	309
Using a While Loop:	309
Resulting Output	310
Directed UDP Broadcast	311
Visual Basic	311
Java	311
C++	312
Connecting to a Specific Node, Creating Enterprise Users, Login, Logout	312
Visual Basic	312
Java	315
C++	318
Resulting Output	321
Working with Staffware Lists — SWLists, SWViews, and SWLocLists	322
Visual Basic	322
Java	326
C++	331
Resulting Output	336
Working with Staffware Lists — SWXLists	339
Visual Basic	339
Java	343
C++	348
Resulting Output	353
Index	357

Preface

Introduction

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

This document provides descriptions of the functionality of TIBCO iProcess Objects. For details about the syntax for the properties and methods used to provide that functionality, see the on-line help provided with your TIBCO iProcess Objects.

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

For the latest TIBCO Staffware Process Suite 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) COM Client	TIBCO iProcess™ Objects (COM) ^a
Staffware Process Objects (SPO) Java Client	TIBCO iProcess™ Objects (Java) ^a
Staffware Process Objects (SPO) C++ Client	TIBCO iProcess™ Objects (C++) ^a
Staffware Process Objects (SPO) Server	TIBCO iProcess™ Objects Server
Staffware Process Objects (SPO) Director	TIBCO iProcess™ Objects Director
Staffware iProcess Engine	TIBCO iProcess™ Engine
Staffware Process Definer (SPD)	TIBCO iProcess™ Modeler

a. Collectively, these are referred to as the "TIBCO iProcess Objects".

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 and SPO in the software (e.g., file and directory names) and documentation.

Knowledge Level

The intended audience of this document is programmers and technical consultants who are using the objects that comprise TIBCO iProcess Objects 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 Objects On-Line Help** - This provides syntax information for all objects, properties, and methods available to programmers.
- **TIBCO iProcess Objects Object Model Graphic** - This provides a graphical representation of the objects that comprise the TIBCO iProcess Objects.
- **TIBCO iProcess Objects Installation Guide** - This guide provides the steps you need to follow to successfully install your TIBCO iProcess Objects software.
- **TIBCO iProcess Objects Release Notes** - The release notes provide information about changes that have occurred in each release of TIBCO iProcess Objects. 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 Objects. 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 the TIBCO iProcess Objects	Summary of Changes
Issue 1	Mar. 2002	9.0(0.0)	Initial issue
Issue 2	Mar. 2002	9.0(0.0)	Added “SPO” to the title of the document. Also includes minor additions/corrections.
Issue 3	Feb. 2003	10.0(0.0)	New filtering information added to <i>Filtering Work Items and Cases</i> chapter. Also miscellaneous changes throughout.
Issue 4	Mar. 2003	10.0(0.0)	Added <i>Procedures</i> chapter. Added sections for predicting cases, suspending cases, jumping to new outstanding step, using graft steps, plus miscellaneous corrections.
Issue 5	May 2003	10.0(0.0)	The <i>Filtering Work Items and Cases</i> chapter was split into three separate chapters. The reader will use the appropriate chapter, depending on which filtering enhancements have been incorporated into their TIBCO iProcess Objects Server.
Issue 6	Sep. 2003	10.0(1.0)	Added information about the new outstanding item objects (SWEventStep, SWGraftStep, etc.), sub-proc precedence on StartCase, plus miscellaneous changes throughout.
Issue 7	Sep. 2003	10.0(1.0)	Removed references to the TIBCO iProcess Objects Director (as it is not available yet).
Issue 8	Dec. 2003	10.0(2.0)	Added information concerning running multiple instances of the TIBCO iProcess Objects Server and using in-memory logging.
Issue 9	June 2004	10.0(4.2)	Additional information added about running multiple instances of the TIBCO iProcess Objects Server on Windows (configuration utility changes), MoveSysInfo method, AddNodeEx and MakeNodeInfoEx methods, MaxCnt property, etc.
Issue 10	Feb. 2005	10.2.0	Added information about transaction control steps, TIBCO iProcess Objects Director, as well as new methods used to specify specific case fields, CDQPs, and markings when creating, locking, and accessing work items (MakeWorkItemEx, MakeWorkItemByTagEx, MakeXListItemsEx, LockItemMarkings, LockItemsMarkings, and CDQPNames).
n/a	June 2005	10.2.1	Updated information about the TIBCO iProcess Objects Director to reflect changes made during testing of the software. 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 at time of issue.

Issue	Date	Current Version of the TIBCO iProcess Objects	Summary of Changes
n/a	Oct. 2005	10.3.0	Added information about activity publication and database configuration access. Removed information about the TIBCO iProcess Objects Server (starting/stopping, configuration parameters, logging, etc.) — for this information, see the <i>TIBCO iProcess Objects Server Administrator's Guide</i> . Also removed information about the TIBCO iProcess Objects Director — for this information, see the <i>TIBCO iProcess Objects Director Administrator's Guide</i> .
n/a	May 2010	10.4.0	Added UTF-8 support on all platforms.

Introduction

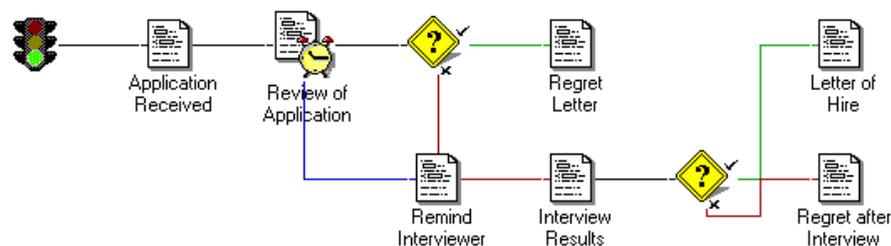
Introduction

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

The objects in the 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

A business process that is automated with TIBCO software is referred to as a “procedure.” Procedures are defined with a TIBCO tool called the TIBCO iProcess Modeler. 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 Procedure

Before describing the underlying architecture of TIBCO iProcess Objects, 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 the TIBCO iProcess Modeler. An example is shown in the illustration above.
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.

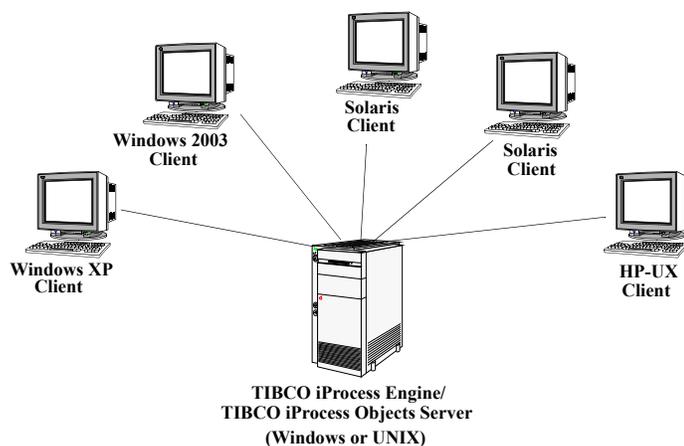
Term	Definition
Step	A procedure is made up of a number of steps, which define the activities that take place within the flow of a procedure. Each step defines what must be done, who must do it, and, optionally, a deadline by which it must be done.
Work Queue	This is a list of work items (see below) 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 one of the action items 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, giving that user access privileges to log in to that node. Each user has a work queue that has the same name as the user name given that person when they were 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 name given that group when it was 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 Objects, allowing client applications to make use of that information and functionality in the business processes they automate.

TIBCO iProcess Objects Configurations

TIBCO iProcess Objects systems are designed to operate in a client/server configuration. Client applications making use of objects exposed by TIBCO iProcess Objects communicate via TCP/IP to a TIBCO iProcess Objects Server. The TIBCO iProcess Objects Server acts as a gateway to a TIBCO iProcess Engine, which manages work queues and case data for the client application.

Clients and Servers can run on a variety of platforms, which can be mixed and matched in any configuration (see the following illustration).



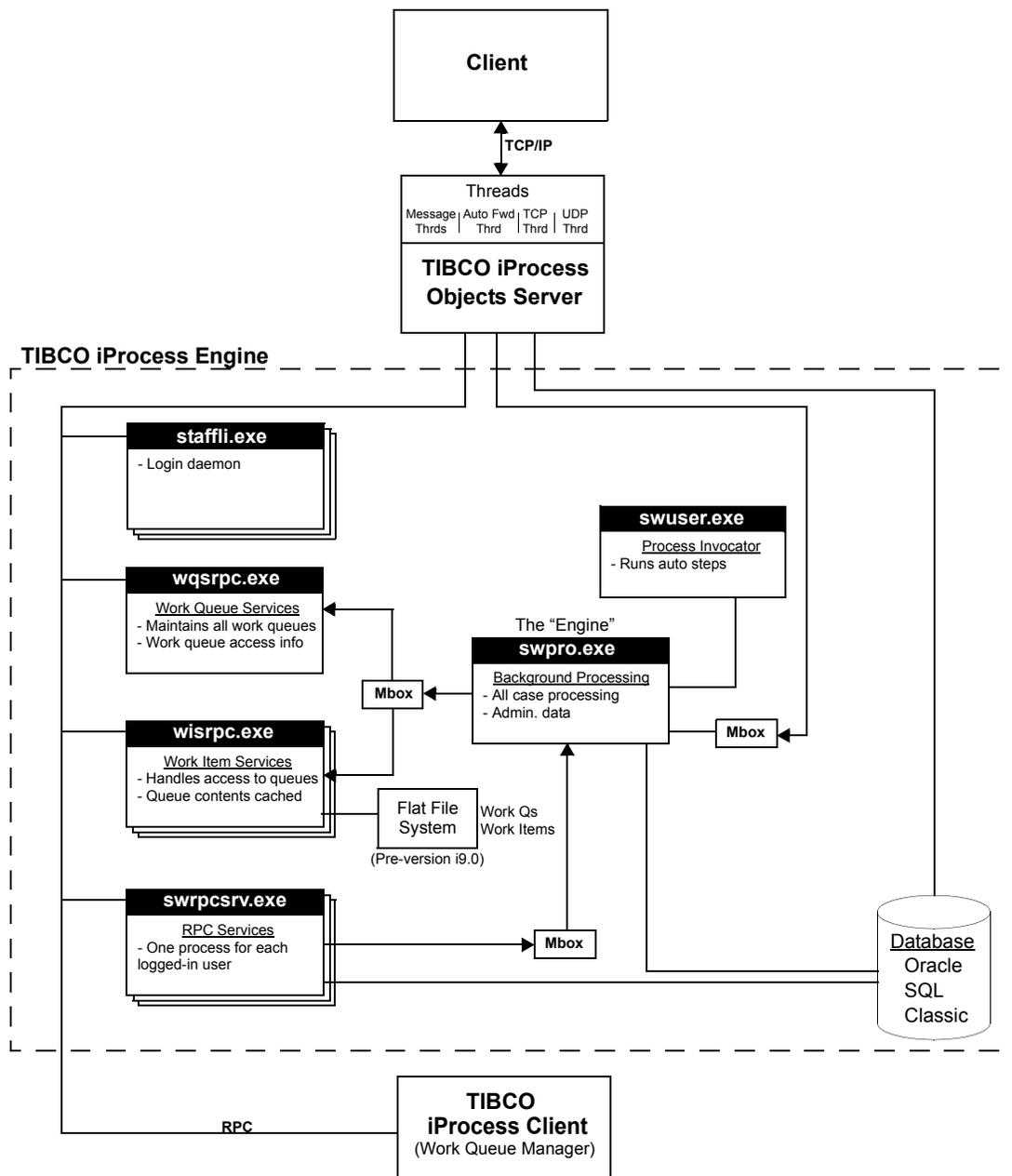
TIBCO iProcess Objects Configuration

The platforms currently supported are as follows:

- **TIBCO iProcess Objects (COM):**
 - Windows XP
 - Windows Server 2003
- **TIBCO iProcess Objects (Java) and TIBCO iProcess Objects (C++):**
 - Windows XP
 - Windows Server 2003
 - Solaris
 - HP-UX
 - AIX
- **TIBCO iProcess Objects Server:**
 - Windows XP
 - Windows Server 2003
 - Sun Solaris
 - HP-UX
 - IBM AIX
 - Linux

TIBCO iProcess Objects Components

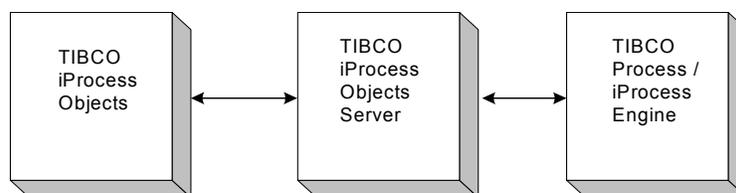
The diagram below illustrates how particular components of a TIBCO iProcess Objects system interact in a Windows environment.



The following subsections describe the components in a typical installation.

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 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 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 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 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 `$WDIR\swdefs` (Windows) or `$$WDIR/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 **swadmin** users.

Engine Processes

There are a number of processes that run on the engine that perform the tasks necessary to process cases. They include:

- **Background Process** (**swpro.exe** in Windows; **nodename.bkg** in UNIX) - Retrieves, routes, and processes data according to the instruction set received through the mbox file (for information about the mbox file, see the *TIBCO iProcess Objects Server Administrator's Guide*).

A special user account by the name of **swpro** must exist in the O/S. This account is used by the TIBCO iProcess Engine to run the background process. The **swpro** user must be an administrator and needs the advanced user right of "Act as part of the operating system." This user must also be a member of the Staffware Users Group.

- **Work Queue Server** (**wqsrpc.exe** in Windows; **wqsrpc** in UNIX) - Handles all of the user and group work queues, and controls access to those queues. There is only a single **wqsrpc** process running at any time for a TIBCO iProcess Engine.
- **Work Item Server** (**wisrpc.exe** in Windows; **wisrpc** in UNIX) - Controls all of the work items in the work queues, and maintains a cache of all information in those queues. There can be multiple **wisrpc** processes running at one time, with five as the default. (This is configurable with the `WQS_WIS_COUNT` parameter in the `SWDIR\etc\staffcfg` file.)
- **RPC Server** (**swrpcsrv.exe** in Windows; **swrpcsrv** in UNIX) - Monitors Remote Procedure Calls (RPC) on the TIBCO iProcess Engine.
- **Login Daemon** (**staffli.exe** in Windows; **Pstaffli** in UNIX) - Monitors and controls the number of licensed users allowed to login to the TIBCO iProcess Engine.
- **Process Invocator** (**swuser.exe**) - The process that runs auto-steps (execution of external programs) from within procedures.

A special user account by the name of **swuser** is used to run the Process Invocator service. This user must be a member of the Staffware Users Group.

Each of these processes run as a service on Windows, and a daemon on UNIX systems.

Also notice in the illustration on [page 4](#) that the Work Item Server is linked to the *flat file system*. All work queue data (i.e., which work items are in each of the user and group work queues) is stored in flat files (`SWDIR\queues\user\staffo`), one for each user and group on the system. The Work Item Server maintains a cache of this information and uses it to present work items on the work queues. All other data (procedure definitions, user/group definitions, field definitions, case data, etc.), is stored in the database (Oracle, SQL, or Classic). (*Note - If you are using a TIBCO iProcess Engine, all data is stored in database tables rather than in flat files. For information about these tables, see the TIBCO iProcess Engine Administrator's Guide.*)

The **TIBCO iProcess Client** is an out-of-the-box client application that allows the viewing and processing of work items. The TIBCO iProcess Client does not use TIBCO iProcess Objects. It talks directly to the TIBCO iProcess Engine (as shown in the illustration on [page 4](#)). Client applications developed with TIBCO iProcess Objects, and the TIBCO iProcess Client, can perform operations that affect work queues and work items when viewed by the other application.

TIBCO iProcess Objects

TIBCO iProcess Objects expose the interfaces of the objects provided in the object model. TIBCO iProcess Objects make requests to the TIBCO iProcess Objects Server on behalf of the controlling application to access and control data on the TIBCO iProcess Engine. Communication between the TIBCO iProcess Objects and the TIBCO iProcess Objects Server is via direct TCP/IP sockets.

The publicly exposed objects in the TIBCO iProcess Objects are available in a number of flavors:

- **TIBCO iProcess Objects (COM)** - The TIBCO iProcess Objects (COM) can be used in any COM container (Visual Basic, ASP, etc.). TIBCO iProcess Objects are available for Windows XP and 2003. They are implemented as an in-process DLL, sharing process space with the client application. Therefore, each client application that uses TIBCO iProcess Objects (COM) will have a copy of the DLL as part of its process address space.
- **TIBCO iProcess Objects (Java)** - The TIBCO iProcess Objects (Java) is based on Java bindings wrapping C++ classes. You can use any appropriate Java development tool to access the objects via these Java classes. The TIBCO iProcess Objects (Java) are available for Windows XP and 2003, as well as three versions of UNIX (HP-UX, IBM AIX, and Sun Solaris).
- **TIBCO iProcess Objects (C++)** - The TIBCO iProcess Objects (C++) provide public C++ classes. Development with this product can be performed with any C++ development tool, such as Visual C++. This client is available for Windows XP and 2003, as well as three versions of UNIX (HP-UX, IBM AIX, and Sun Solaris).

Client-to-Server Communications

TIBCO iProcess Objects communicate with TIBCO iProcess Objects Servers over TCP/IP. You have a number of configuration options for establishing this communication. These options are described in detail in [“Accessing Nodes on the Network” on page 20](#).

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*.

JVM Finalization and Garbage Collection

This section is only applicable to TIBCO iProcess Objects (Java).

Because of a deficiency in how the Java Virtual Machine (JVM) handles finalization and garbage collection, Java Finalization Monitoring classes have been added to TIBCO iProcess Objects.

TIBCO iProcess Objects (Java) objects require finalization because they use JNI to wrap native C code objects. Finalization provides the mechanism to free the system memory held by the C objects once the wrapping Java objects are no longer being referenced.

The finalization problem arises when the JVM is operating under a heavy load (i.e., many threads concurrently creating large numbers of objects). The rate at which objects are finalized does not keep up with the rate at which objects are freed to be garbage collected. The **finalize** method of these objects must run before:

- The C memory held by the object is released.
- The objects can be garbage collected.

As a result, both the JVM heap memory and the system memory used on the C side continue to grow until one or the other runs out of memory. Which runs out of memory first depends on the JVM heap size and the available system memory. In either case, the process fails with an out of memory condition.

The JVM settings that control garbage collection behavior do not provide a comprehensive way of controlling this problem. These settings can be adjusted in ways that prolong the length of time before the process fails, but extensive testing clearly shows that garbage collection can fall behind on finalization under heavy loads regardless of the JVM settings used.

Java provides the following two methods that can be used in conjunction with garbage collection.

- **System.runFinalization()**
- **System.gc()**

These methods, however, are not deterministic as to what extent they will satisfy the request. In practice, calls to `System.runFinalization()` are not effective under load because the thread that runs finalization is not given enough processing time to keep up with the demand. Calls to `System.gc()` are expensive and serve little purpose when dereferenced objects cannot be reclaimed because they have not yet been finalized.

Solution Strategy

Several strategies are combined in order to overcome this problem with minimum impact on overall process throughput:

- The number of objects created and finalized is monitored.
- The priority of the finalization thread is set to the maximum value.
- A range is specified (`minThreshold`, `maxThreshold`) for the number of objects that are pending finalization (i.e., the difference between the number created and the number finalized). Within this range, actions are initiated to ensure finalization does not fall behind.
- Impact on throughput is kept to a minimum by applying controls to program threads only when needed based on the `minThreshold` / `maxThreshold` range specified.
- Program threads that create new objects while the finalization thread is running are induced to yield more control to the finalization thread by introducing a short `Thread.sleep()` delay.
- If the load increases to a point where the object count is near the maximum threshold value, program threads are forced to yield control to the finalization thread by blocking their actions until finalization can bring the object count back down below the maximum level.

Implementation

The `FinalizeMonitor` class implements the strategy for this solution. An additional class, `FinalizePrioritySetter`, is used to set the finalization thread priority. Both classes are in the `SWEntObj` package. `FinalizeMonitor` is a singleton with two public static methods used in tracking objects created and objects finalized. These are:

- **`FinalizeMonitor.incCreated()`**
- **`FinalizeMonitor.incFinalized()`**

Each Java class with a `finalize()` method calls `incCreated()` in each of its constructor methods and `incFinalized()` in its `finalize()` method.

`FinalizeMonitor` is initialized with one of the following methods:

- **`setMaxThreshold(int maxThreshold)`** (default `minThreshold` = 60% `maxThreshold`)
- **`setThresholdRange(int minThreshold, int maxThreshold)`**

Based on the threshold range specified, a modulus value is set that controls how frequently checks are made. On each call to `incCreated()`, the number of objects pending finalization is compared to the modulus value. On hitting the modulus value, a separate thread (started by the `FinalizeMonitor`) is notified, which performs the following actions:

- Compares number of objects pending finalization to the `minThreshold` value. If below this threshold, no actions are taken.
- Compares number of objects pending finalization to a `bypassThreshold` value. This value is a limit near the `maxThreshold` value (`maxThreshold` - modulus). At object counts below this value, program threads creating new objects are subject to a 5 msec delay. Above this value, program threads creating new objects are blocked until the object count falls below the `maxThreshold` value.
- `System.runFinalization` is called.

- If the count has reached the `maxThreshold` value, a loop is entered that calls `System.runFinalization` up to four times and then calls `System.gc()` if required. This loop exits when the count falls below the `maxThreshold` value.

Note that if you are running the `FinalizeMonitor`, you must explicitly stop the `FinalizeMonitor` thread before stopping the client application. If the `FinalizeMonitor` is still running when the client application is stopped, the client application will not exit as expected. To explicitly stop the `FinalizeMonitor` thread, use the following static method:

- **`FinalizeMonitor.stop()`**

You can also use the following static method to determine the current operating status of the `FinalizeMonitor`:

- **`FinalizeMonitor.checkThreadStatus()`**

This method returns one of the following statuses: “Not running”, “Running”, or “Stopped”.

Configuration

- **JVM Heap Size**

In order to make effective use of available memory, the JVM heap size needs to be balanced with the remaining system memory needed by the C objects. The system memory used by the C objects is proportionally higher compared to the memory used by the Java wrapper objects. A 10:1 ratio is a good starting point, with adjustments made based on the specific application (e.g., `java -Xmx125m` would be appropriate if total available memory = 1.25 GB).

- In Windows, use either the Task Manager or the Performance Monitor (available under Administrative Tools) to monitor the amount of system memory used by the Java process.
- In UNIX, the command-line program, "top", can be used to monitor the amount of system memory used by the Java process (774M in the example top output below):

```
>top
...
Memory: 2048M real, 290M free, 2680M swap in use, 327M swap free

  PID USERNAME  THR PRI NICE  SIZE  RES STATE  TIME  CPU COMMAND
 20451 system    58  59   8  774M  717M cpu/1   5:12 30.72% java
  4810 system    14  34   8   58M   12M sleep 140:38  0.19% rmiregistry
```

- The JVM memory can be monitored using the `-verbose:gc` JVM command-line argument. In the example below, Java was started with:

```
java -Xms125m -Xmx125m -verbose:gc ...
```

This generates system output similar to:

```
Full GC 98057K->64950K(118784K), 3.2819178 secs
```

- **FinalizeMonitor**

- The FinalizeMonitor is initialized with one of the static methods:

```
FinalizeMonitor.setMaxThreshold(int maxThreshold)
```

(default minThreshold = 60% maxThreshold)

```
FinalizeMonitor.setThresholdRange(int minThreshold, int maxThreshold)
```

See the Client Main Example below for an example of a client main method that initializes the FinalizeMonitor using command-line arguments.

- Details on FinalizeMonitor actions can be logged to the standard output using:

```
FinalizeMonitor.setVerbose(true) (default = false)
```

Sample output:

```
event elapsedTime created bypass final pending obj/sec  
FM1 1034.875 2012667 6955 1599219 420403 2750  
event elapsedTime created bypass final pending obj/sec  
FM2 1041.348 2012667 7145 1689112 330700 2750
```

- **Determining the maxThreshold Value**

The maxThreshold value should be set to the maximum number of objects that can be pending finalization without running out of system or JVM memory. This can be determined using the test mode of the FinalizeMonitor. Set the test mode using:

```
FinalizeMonitor.setTestMode(int objectCount)
```

where objectCount specifies the number of objects created between each log message output.

In the test mode, the FinalizeMonitor finalization actions are disabled, but the number of objects created, finalized, and pending finalization is tracked and this is logged to the standard output with data similar to the example below:

```
event elapsedTime created bypass final pending obj/sec  
FMTest 1782.1 RMI_TCP_Conn17 2030001 1404522 625479 2612
```

The test should be run under load (i.e., many threads concurrently creating large numbers of objects) that would be similar to the maximum operational loads expected.

The count of objects pending finalization will continue to grow until either the system or JVM memory is exceeded. The maxThreshold value should be set to no more than 85% of the number of objects pending finalization just before running out of memory. In the example above, the objects pending finalization = 625479. A setting of approximately 85% of this (maxThreshold = 530000) would be a good initial setting.

The `minThreshold` setting can be set to any value greater than zero and less than or equal to the `maxThreshold` value. This affects when the `FinalizeMonitor` begins taking actions. The best setting will depend on the specific operational characteristics of the running system:

- A lower setting would result in finalization running more frequently and for shorter durations.
If a system typically has a relatively small number of objects referenced at a given time, a lower `minThreshold` value would potentially free memory sooner.
- A higher setting would result in finalization running less frequently and for longer durations.
If a system typically has a relatively large number of objects referenced at a given time, a higher `minThreshold` value would delay attempts at running finalization until a point where it is more likely that objects would be available to be finalized.

• Client Main Example

A client main method could initialize the `FinalizeMonitor` using command-line arguments similar to the example below:

```
public static void main(String[] args) {
...
    int minThreshold = -1;
    for (int i = 0; i < args.length; i++) {
        String arg = args[i];
        if (arg.equals("-verbose:fm")) {
            FinalizeMonitor.setVerbose(true);
        }
        if (arg.length() > 14 && arg.substring(0, 14).equals("testModeCount=")) {
            FinalizeMonitor.setTestMode(Integer.parseInt(arg.substring(14)));
        }
        if (arg.length() > 13 && arg.substring(0, 13).equals("minThreshold=")) {
            minThreshold = Integer.parseInt(arg.substring(13));
        }
        if (arg.length() > 13 && arg.substring(0, 13).equals("maxThreshold=")) {
            int maxThreshold = Integer.parseInt(arg.substring(13));
            if (minThreshold != -1) {
                FinalizeMonitor.setThresholdRange(minThreshold, maxThreshold);
            }
            else {
                FinalizeMonitor.setMaxThreshold(maxThreshold);
            }
        }
    }
...
}
```

FinalizeMonitor Log Example

The following example shows a typical log output from the FinalizeMonitor:

```
event elapsedTime created bypass final pending obj/sec
FM1 39.212 45018 12 37521 7509 1148
FMEX 39.212
```

where:

- event: This represents an event and will be one of the following:
 - FMTest - Indicates running in test mode. Information logged, but no actions taken.
 - FM1 - Starting a finalization check.

If action is taken to do finalization (i.e., pending \geq check threshold). Note: # indicates a loop count:

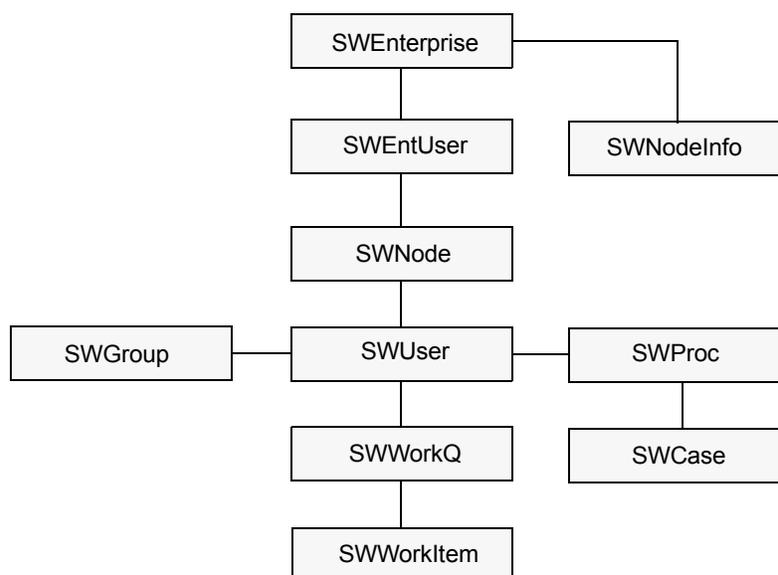
- FMFB# - Before call to induce finalization. (if required)
- FMFA# - After call to induce finalization. (if required)
- FMGB# - Before call to do Garbage Collection. (if required)
- FMGA# - After call to do Garbage Collection. (if required)
- FM2 - Logged at completion of a check if action is taken to do finalization.
- FMEX - Logged when exiting a finalization check if no action taken.
- << - This is a line resulting from a call to printStatus().
- elapsedTime: The number of seconds the process has been running.
- created: The total number of objects created (less those created by bypass).
 - created + bypass (all objects created) = final + pending
- bypass: These are objects created by threads while the checkThread is active.
- final: The number of objects that have been finalized.
- pending: The number of objects that have not been finalized.
- obj/sec: The overall average number of objects created / second (total created + bypass / elapsed-Time)

2

The Object Model

Introduction

The TIBCO iProcess Objects object model defines the hierarchy and relationship between the objects that comprise TIBCO iProcess Objects. The illustration below shows the relationship between some of the primary objects. (Note that there are many other objects that perform secondary roles to those shown in the illustration — the objects shown here, however, illustrate the primary flow of information through the object model.)



Note - An “object model graphic” that shows the entire object model is included on the TIBCO iProcess Objects distribution CD. Refer to that graphic for a comprehensive illustration of the object model for your specific TIBCO iProcess Objects (COM, Java, or C++).

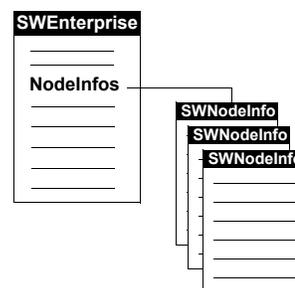
A summary of each of the primary objects shown in the illustration is provided below.

- **SWEnterprise** - This object is the root of the object model. It provides an enterprise-wide view of the entire installation. SWEnterprise is one of only four objects that can be directly created (the others are SWMarking, SWSortField, and SWDate). All other objects in the object model are internally instantiated and returned by other objects.

See [“Creating the SWEnterprise Object” on page 19](#) for information about how to create the SWEnterprise object.

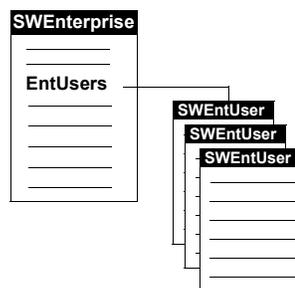
- **SWNodeInfo** - This object contains information about the nodes (TIBCO iProcess Objects Servers) that are available for the client to communicate with. There will be one SWNodeInfo object for each available TIBCO iProcess Objects Server. This list of SWNodeInfo objects is stored in the **NodeInfos** property on SWEnterprise.

For information about how the client determines available TIBCO iProcess Objects Servers, see [“Accessing Nodes on the Network”](#) on page 20.



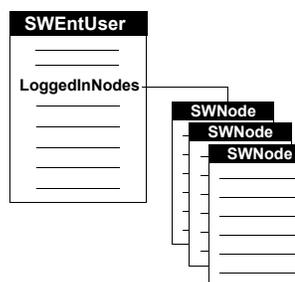
- **SWEntUser** - This object represents an *Enterprise User*. The enterprise user object conceptually represents a single Staffware user throughout the enterprise (across multiple nodes). This object contains information about a Staffware user in the enterprise, including common logon information and the nodes that the user is logged onto in the enterprise. The list of SWEntUser objects is stored in the **EntUsers** property on SWEnterprise.

For more information about the SWEntUser object, see [“Creating Enterprise Users”](#) on page 31.



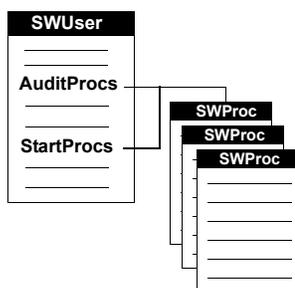
- **SWNode** - This object represents a specific node (TIBCO iProcess Objects Server) in the enterprise. The SWNode object contains information about the procedures, users, groups, work queues, etc., that belong to the node. For every node that an enterprise user is logged into, one SWNode object is added to the list of SWNode objects in the **LoggedInNodes** property on the SWEntUser object.

For more information, see [“Logging In”](#) on page 32.

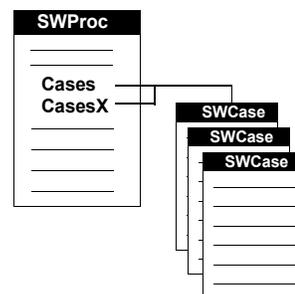


- **SWProc** - This object represents a procedure. It contains methods that are used to start, close, and purge cases (instances of procedures), and properties that contain information about the procedure. The **AuditProcs** and **StartProcs** properties on SWUser contain lists of procedures (SWProc objects) that the logged in user has permission to audit and start.

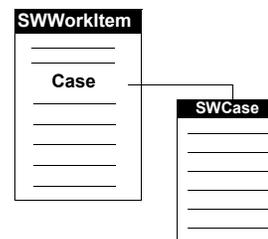
For information about starting a case of a procedure, see [“Starting a Case”](#) on page 230.



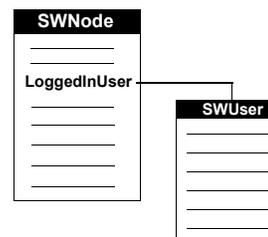
- **SWCase** - This object represents a specific instance of a procedure. It contains information such as a unique case reference number, the name of the procedure for which it was started, who started it, etc. It also contains audit information, which is used to track the progress through the steps in the case. The SWProc object contains two properties that return a list of the currently active cases for that procedure: one is the **Cases** property, which returns a “view” of SWCase objects, and the other is the **CasesX** property, which returns an “XList” of SWCase objects (for information about views and XLists, see the *Working with Lists* chapter on [page 55](#)).



The SWWorkItem object has a **Case** property that contains the SWCase object that represents the case to which the work item belongs.

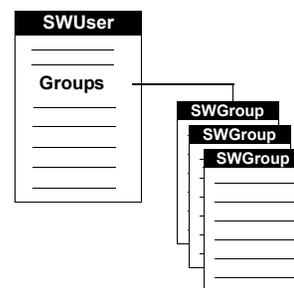


- **SWUser** - This object represents a Staffware user. It contains properties that define things such as the procedures the user can start, groups to which the user belongs, etc. The **LoggedInUser** property on SWNode contains an SWUser object for the person logged into the node.



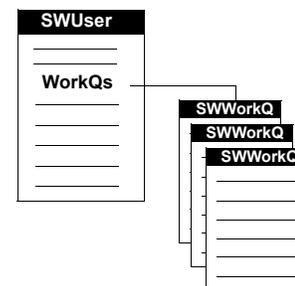
For more information, see [“User Administration” on page 214](#).

- **SWGroup** - This object represents a collection of users. The **Groups** property on SWUser contains a list of the groups (SWGroup objects) to which the user belongs.



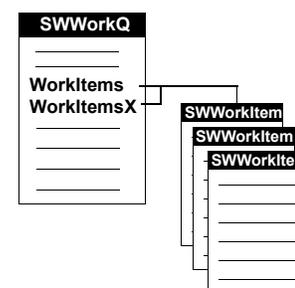
For more information, see [“User Administration” on page 214](#).

- **SWWorkQ** - This object represents either a personal work queue (one owned by an individual user) or a group work queue (one that is shared among several users). It contains methods that are used to process (lock, keep, release, etc.) multiple work items that are in the work queue. The **WorkQs** property on the SWUser object contains a list of the work queues to which the user has access, i.e., the work queues on which he will perform work.



Note that the list of work queues available from the SWNode object are used for administrative and reporting purposes.

- **SWWorkItem** - This object represents a single work item that is listed in a work queue, awaiting some sort of action by a user. It contains many Boolean properties that indicate the work item’s current state (e.g., locked, unopened, urgent, etc.). It also provides methods that are used to process (lock, keep, release, etc.) this particular work item. The SWWorkQ object contains two properties that return a list of work items that are currently in that work queue: one is the **WorkItems** property, which returns a “view” of SWWorkItem objects, and the other is the **WorkItemsX** property, which returns an “XList” of SWWorkItem objects (for information about views and XLists, see the *Working with Lists* chapter on [page 55](#)).



Naming Conventions

Overview

The following naming conventions have been incorporated into the TIBCO iProcess Objects:

- If the property or method name ends in “**Cnt**”, it is a counter and will be returned as an integer or a long, depending on the expected maximum size of the value.
- If the property or method name ends in “**Names**”, it is a list of strings rather than a list of objects.
- If the property or method name ends in an “**s**” (but not “**Names**”), it is a List, Local List, View, or XList containing objects. The name of the property/method also tells you the name of the object that is returned when you invoke the property/method. For example, **Groups** returns **SWGroup** objects; **Fields** returns **SWField** objects.
- If the property or method name ends in an “**X**”, it is an XList containing objects.
- If the property or method name begins with “**Is**”, it is a boolean value.
- If the property/method name ends in an “**Ex**”, it means one of two things:
 - It is a property or method that provides “extended” functionality. The property/method names that use this convention are StartCaseEx and NodeInfoEx (which returns an SWNodeInfoEx object).
 - It is a method that performs actions upon work items in an XList (LockItemsEx, KeepItemsEx, etc.).

Differences between COM, Java, and C++ Naming Conventions

Note that this document primarily uses the COM convention of naming items. For instance, it mentions an object’s “properties.” This is specific to COM — objects in Java and C++ don’t have properties — they only have “methods.”

Also, in COM, some properties have read/write attributes, so there is a single property that you can use to read or set the value. In Java and C++, methods don’t have read/write attributes — instead they have corresponding “get” and “set” methods to “read” and “write” the value, respectively. The following are a few examples of this:

COM Property	Java/C++ Methods
PollCnt	getPollCnt / setPollCnt
FieldName	getFieldName / setFieldName

Boolean properties/methods are an exception to the above naming convention. For a COM Boolean property that is readable and writable, again there is a single property (beginning with “Is”). In Java and C++, the method that “reads” the Boolean value begins with “is” (lowercase); the method that is used to “write” the value begins with “set”. Some examples:

COM Property	Java/C++ Methods
IsRebuildAll	isRebuildAll / setRebuildAll
IsWaitForAll	isWaitForAll / setWaitForAll

Looking on the object model graphics provided on the distribution CD will show how these naming conventions were used throughout all three of the object models (COM, Java, and C++).

Getting Started

Creating the SWEnterprise Object

The first thing that needs to be done in a TIBCO iProcess Objects client application is to create the **SWEnterprise** object — the root of the object hierarchy. The SWEnterprise object provides an enterprise-wide view of the entire installation. The connection information for all nodes (TIBCO iProcess Objects Servers) in the enterprise is stored in the SWEnterprise object. It is created in the following manner:

COM Application

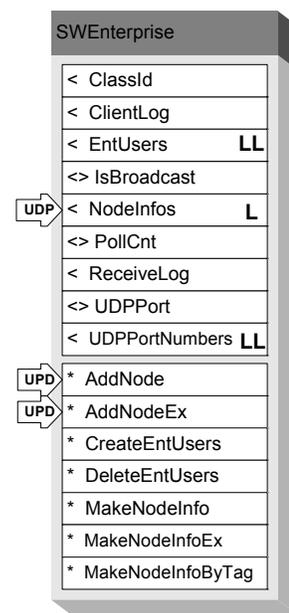
```
Dim oEnterprise As SWEnterprise
Set oEnterprise = New SWEnterprise
```

C++ Application

```
SWEnterprise *pEnterprise;
pEnterprise = new SWEnterprise;
```

Java Application

```
SWEnterprise oEnterprise;
oEnterprise = new SWEnterprise();
```



Do I Create One or Many SWEnterprise Objects?

Typically, in a thick client application you'll only create one SWEnterprise object, and its object handle is stored in a global variable for use throughout the application. However, if multiple threads are being used, it is recommended that an SWEnterprise object be created for each thread (this is an example of a *broker*). Because objects add state, they are *not* intended to be passed between threads.

In a web-based environment, a new SWEnterprise object will need to be created for each web page (typically the state is not saved between pages because of performance reasons). For information about the "stateless" objects, see the *Stateless Programming* chapter on [page 270](#).

When Should the SWEnterprise Object be Destroyed?

Since nearly all other objects are dependent on the SWEnterprise object, it should not be deleted (destroyed) until the client application no longer needs to access data on the TIBCO iProcess Engine. If the SWEnterprise object is deleted prior to the destruction of other objects, the other objects can continue to be used, but no new logins can be done since the SWEnterprise object holds the connection information about the nodes available in the enterprise.

Accessing Nodes on the Network

Once the SWEnterprise object is created, you need to determine which TIBCO iProcess Objects Servers are available on the network so that you can connect to one of them. This can be done in the following ways:

- **Automatic Discovery (UDP Broadcast) on the LAN Segment** - This is the default method of finding nodes on the network. It is typically used when you don't know any information about the nodes that are available. One limitation of this method is that UDP broadcasts will not go across routers. See [“Automatic Discovery \(UDP Broadcast\) on a LAN Segment” on page 22](#) for details of using this method.
- **Directed UDP to a Specific Node** - This method (using either the `AddNode` or `AddNodeEx` method) employs a directed UDP message. It can be used if you know the name or IP address of the machine running the TIBCO iProcess Objects Server of interest. This method can be used across routers. See [“Directed UDP to a Specific Node” on page 25](#) for details of using this method.
- **Manually Create an SWNodeInfo Object**- This method (using either the `MakeNodeInfo` or `MakeNodeInfoEx` method) requires that you know everything about the node — computer name, node name, IP address, and TCP port number. This is the method that you would use in a web-based/ASP environment. See [“Manually Create an SWNodeInfo Object” on page 26](#) for details of using this method. (Note - The `MakeNodeInfoByTag` method is also available — see [“Stateless Programming” on page 270](#) for more information.)
- **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 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 SWNodeInfo objects and sending UDP broadcasts/messages apply to both TIBCO iProcess Objects Servers and TIBCO iProcess Objects Directors; clients obtain an SWNodeInfo object for a TIBCO iProcess Objects Director in the same way as for a TIBCO iProcess Objects Server.

These methods are not mutually exclusive — depending upon the architecture of your system, you may choose to use one, two, or even all three methods to access nodes.

Some factors you need to look at when deciding the method to use for accessing nodes are:

- Length of the login
- Whether you are on a LAN or a WAN
- How much information you know about the available nodes
- Verification of the node

These factors are described in the subsections that follow. Following these descriptions is a table that summarizes the choices you have depending on various factors.

Length of the Login

A major deciding factor in which method(s) you use is whether your logins are *long-lived* (thick client, where logins are typically of a long duration — hours, or even days in length) or *short-lived* (web-based client, where logins are commonly very short — sometimes only seconds, or minutes in length).

Latency is a prime concern when you are working in a web-based environment (short-lived logins) because a login occurs on every web page access. It isn't a concern in thick clients (long-lived logins) because the user will typically login once and be around for a long duration.

This factor also dictates how the TCP port on the TIBCO iProcess Objects Server is configured — as *static* or *dynamic* (see [“Configuring the TIBCO iProcess Objects Server TCP Port” on page 27](#) for information about how to configure the TIBCO iProcess Objects Server TCP port).

- **Short-lived** - This type of login implies frequent logins such as in a web-based environment where it is desirable to login on each web page. In this environment, you can't afford much latency, therefore a UDP broadcast or directed UDP can't be used. The “manually create an SWNodeInfo object” method (MakeNodeInfo or MakeNodeInfoEx) must be used instead.

Since you are using MakeNodeInfo / MakeNodeInfoEx with a short-lived login, the TIBCO iProcess Objects Server will need to be configured with a static TCP port so you know what port number to use with the MakeNodeInfo/MakeNodeInfoEx method.

- **Long-lived** - This login implies a thick client. In other words, the user is using a client-side application where the user may stay logged in for long durations, such as hours or all day. In this environment, latency is not an issue, therefore you can use any one of the three accessing methods.

With a long-lived login, you can configure the TCP port either dynamic or static.

- **Both Long-lived and Short-lived** - This is an environment where you have both web-based clients and thick clients using the same TIBCO iProcess Objects Server. The web-based clients will use the “manually create an SWNodeInfo object” method (MakeNodeInfo/MakeNodeInfoEx), and the thick clients can use any one of the three methods.

In this mixed environment, the TIBCO iProcess Objects Server will need to be configured with a static TCP port so the web-based clients can specify the TCP port number in the MakeNodeInfo/MakeNodeInfoEx method. The TIBCO iProcess Objects Server will still respond to UDP broadcasts and directed UDPs with the static TCP port number.

LAN or WAN

- **LAN** - Across a LAN, you can use any of the three accessing methods. The auto-discovery UDP broadcast can be used across a LAN segment.
- **WAN** - The auto-discovery UDP broadcast may not be able to be used across a WAN, therefore, in this environment, you should use either the directed UDP (AddNode/AddNodeEx method) or the “manually create an SWNodeInfo object” method (MakeNodeInfo/MakeNodeInfoEx).

Amount of Information you know about the Available Nodes

Which method(s) you choose may be determined by the amount of information you know about the available nodes — with the UDP broadcast requiring the least amount of information and the “manually create an SWNodeInfo object” method requiring the most.

- **Auto-discovery** (UDP) broadcast doesn't require that you know anything about the available nodes — the nodes that are available will respond.
- **Directed UDP** requires that you know the IP address or computer name of the node.
- The **Manually Create an SWNodeInfo Object** method requires that you know the computer name, node name, IP address, and TCP port number of the node.

Verification of the Node

If a verification of the node is required, you must use either the UDP broadcast or directed UDP. The “manually create an SWNodeInfo object” method does not verify the validity of the information passed in the MakeNodeInfo/MakeNodeInfoEx method until the login method on SWEntuser is called. If any of the information provided is incorrect, an swWINSOCKErr is returned.

The table below shows the access methods that are possible based on various criteria.

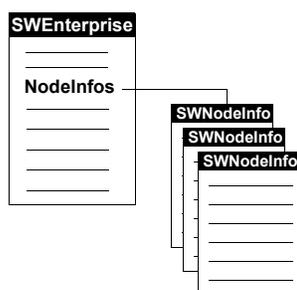
	LAN	WAN	Web App	Broker	Requires computer name or IP address	Requires TCP port number	Verify server info
UDP Broadcast	X						X
Directed UDP	X	X		X	X		X
Manually Create an SWNodeInfo Object	X	X	X	X	X	X	

Automatic Discovery (UDP Broadcast) on a LAN Segment

This is the default method of determining the TIBCO iProcess Objects Servers that are available on the network. Note that if you are using a TIBCO iProcess Objects Director to choose a TIBCO iProcess Objects Server for you, this mechanism can also be used to determine the available TIBCO iProcess Objects Directors.

To auto-discover nodes (TIBCO iProcess Objects Servers or TIBCO iProcess Objects Directors), a User Datagram Protocol (UDP) broadcast is issued by the client, requesting all TIBCO iProcess Objects Servers/Directors on the LAN segment to return a message identifying themselves. The information they return is then used to create **SWNodeInfo** objects, one for each node that responds to the broadcast. These SWNodeInfo objects are added to the list in the **NodeInfos** property on SWEnterprise.

The **IsBroadcast** property must be set to true (the default) for UDP broadcast to be enabled.



What actually causes the UDP broadcast to be issued? One of two things:

- Accessing the **NodeInfos** list for the first time after creating an SWEnterprise object, or
- executing the **Rebuild** method on the NodeInfos list.

The NodeInfos list can also be rebuilt (with the **Rebuild** method) at a later time to update the list of TIBCO iProcess Objects Servers/Directors.

Configuring the UDP Broadcast

The SWEnterprise object contains two methods that are used to configure the UDP broadcast — one sets the interval of the UDP broadcast and the other specifies the port number on which the broadcast is issued.

- **Set the Broadcast Interval** - Occasionally, machines that are listening on the network miss the auto-discovery broadcasts. This is the nature of the auto-discovery broadcast mechanism — there is no guarantee that available nodes will hear and respond to the broadcast. To compensate for this, the SWEnterprise object can be configured to set the number of UDP broadcasts the client will issue. That value is held in the **PollCnt** property of SWEnterprise. Broadcasts are made once per second, with the total number of broadcasts equaling the value in PollCnt.

A PollCnt of 2 may be sufficient for many installations — this allows the nodes that don't hear the first broadcast to hear the second one. A PollCnt of 2 may be too small, however, if your LAN experiences a high latency; it may take longer than 2 seconds for the response to return from the node. In this case, the default value of 5 may be a better option.

- **Specify the Broadcast Port Number** - By default, UDP messages are issued by the client on port 55666. You can use the **SWEnterprise.UDPPort** property to specify the port on which UDP messages will be issued.

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 **UDPPortNumbers** property allows for specifying multiple UDP ports. Note that this property is a local list, requiring you to use the **UDPPortNumbers.Add** method to add port numbers to it.

Note - If the UDPPort property value is > 0, and there is a list specified in UDPPortNumbers, the UDP broadcast will be sent to both.

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 SWNodeInfo object returned by an 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 in the **InstanceNumber** property on SWNodeInfo. For TIBCO iProcess Objects Servers that don't support multiple instances, or when only a single instance is running on a machine, the InstanceNumber property 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*.

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 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.
 - Use either the **AddNode/AddNodeEx** or **MakeNodeInfo/MakeNodeInfoEx** method. See [“Directed UDP to a Specific Node” on page 25](#) and [“Manually Create an SWNodeInfo Object” on page 26](#) for information about these methods.
- The broadcast interval may be set too small. Try increasing the interval by setting the **PollCnt** property on SWEnterprise.
- 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. Check the values in the **UDPPort** or **UDP-PortNumbers** properties.
- 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 **UDPPortNumbers** property.

Example — Auto-Discovery UDP Broadcast

See [page 307](#) for a comprehensive example.

```
Dim oEnterprise As SWEnterprise
Dim oNodeInfo As SWNodeInfo

On Error GoTo Err_RunSample

Set oEnterprise = New SWEnterprise
oEnterprise.PollCnt = 2           'Broadcast for 2 sec

For Each oNodeInfo In oEnterprise.NodeInfos
    ' Display names in intermediate window
```

```

    Debug.Print "NodeInfo Key = " & oNodeInfo.Key & vbCrLf _
    & "      IP Address= " & oNodeInfo.IPAddr
Next
Exit Sub

```

Directed UDP to a Specific Node

A directed UDP can be issued to a specific TIBCO iProcess Objects Server to determine if it's available. And if you are using a TIBCO iProcess Objects Director to choose a TIBCO iProcess Objects Server for you, you can also send a directed UDP message to the TIBCO iProcess Objects Director to determine if it's available.

A directed UDP is issued with the **AddNode** or **AddNodeEx** method on SWEnterprise. These methods can be used over a LAN, WAN, or across a router where auto-discovery broadcasts can't be used:

- The **AddNode** and **AddNodeEx** methods allow you to provide either the computer name or the IP address of the machine to which you are directing the UDP message. If you provide the computer name instead of the IP address, the client machine must be configured for TCP name resolution, typically either DNS or local host file.
- The **AddNodeEx** method provides an optional *InstanceNumber* parameter that allows you to specify that the UDP message be directed to a specific instance of the TIBCO iProcess Objects Server/Director when multiple instances of a TIBCO iProcess Objects Server/Director are being run on a single machine (for more information about running multiple instances of the TIBCO iProcess Objects Server, see the *TIBCO iProcess Objects Server Administrator's Guide*; for more information about running multiple instances of the TIBCO iProcess Objects Director, see the *TIBCO iProcess Objects Director Administrator's Guide*). If the *InstanceNumber* is specified, it is added to the **Key** of the SWNodeInfo object that is returned. If the *InstanceNumber* is omitted, it defaults to instance number 1.

*Note - If you are issuing a directed UDP to a specific TIBCO iProcess Objects Server or TIBCO iProcess Objects Director, and you do not want a UDP broadcast to also go out, be sure to set the **IsBroadcast** property on SWEnterprise to False to turn off the default auto-discovery broadcast.*

If there is no reply from the directed UDP, an swWINSOCKErr is returned.

If the AddNode/AddNodeEx method adds an SWNodeInfo object to the NodeInfos list that is already in the list, an swLLDupErr is returned.

Specifying the UDP Port Number

By default, UDP messages are issued by the client on port 55666. You can use the **SWEnterprise.UDPPort** property to specify the port on which UDP messages will be issued.

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 **UDPPortNumbers** property allows for specifying multiple UDP ports. Note that this property is a local list, requiring you to use the **UDPPortNumbers.Add** method to add port numbers to it.

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 SWNodeInfo object returned by an 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 in the **InstanceNumber** property on SWNodeInfo. For TIBCO iProcess Objects Servers that don't support multiple instances, or when only a single instance is running on a machine, the InstanceNumber property 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*.

Example — Directed UDP Broadcast

See [page 311](#) for a comprehensive example.

```
Set oEnterprise = New SWEnterprise
oEnterprise.IsBroadcast = False           'No Broadcast

oEnterprise.AddNode "swdoug2", "doug1" 'add a node
```

Manually Create an SWNodeInfo Object

If you know all of the required information, you can manually create an SWNodeInfo that represents the desired TIBCO iProcess Objects Server. And if you are using a TIBCO iProcess Objects Director to choose a TIBCO iProcess Objects Server for you, you can also manually create an SWNodeInfo object that represents the TIBCO iProcess Objects Director.

The **MakeNodeInfo** or **MakeNodeInfoEx** methods are used to manually create an SWNodeInfo object and add it to the NodeInfos list. These methods require that you know the machine name, the node name of the TIBCO iProcess Objects Server/Director, the IP address, and the TCP port number used by the TIBCO iProcess Objects Server/Director. You must be certain the parameters are correct, as these methods do not verify their validity. If any of them are incorrect, an swLoginFailErr error is returned when the Login method on SWEntUser is called.

This method is used for the following reasons:

- TIBCO iProcess Objects are being used in a web-based environment where the latency involved in a UDP broadcast or a directed UDP is an issue.

- This method of adding an SWNodeInfo object to the NodeInfos list can be used in situations where you want to define TIBCO iProcess Objects Servers/Directors that will be available on the network, even though they may not be available at this particular time.
- This method can also be used if there are multiple TIBCO iProcess Objects Servers/Directors running on a single machine. In this environment, the UDP message may only be delivered to one of the TIBCO iProcess Objects Servers/Directors. To remedy this, you could use the **MakeNodeInfo** or **MakeNodeInfoEx** method to manually add each of the TIBCO iProcess Objects Servers/Directors to the NodeInfos list. (Or you could configure each of the servers with its own UDP port, then issue a UDP broadcast over multiple ports — see “[Configuring the UDP Broadcast](#)” on [page 23](#) for more information.)

When manually creating an SWNodeInfo object, the TIBCO iProcess Objects Server’s/Director’s TCP port must have a static assignment since you are specifying the specific port number in an argument with the MakeNodeInfo / MakeNodeInfoEx method. For information about configuring the TCP port on a TIBCO iProcess Objects Server, see “[Configuring the TIBCO iProcess Objects Server TCP Port](#)” on [page 27](#). For information about configuring the TCP port on a TIBCO iProcess Objects Director, see the **TCP_SERVICE_NAME** process attribute in the *TIBCO iProcess Objects Director Administrator’s Guide*.

The MakeNodeInfoEx method provides an optional *InstanceNumber* parameter that allows you to specify a specific instance of the TIBCO iProcess Objects Server/Director when multiple instances of a TIBCO iProcess Objects Server/Director are being run on a single machine (for more information about running multiple instances of the TIBCO iProcess Objects Server, see the *TIBCO iProcess Objects Server Administrator’s Guide*; for more information about running multiple instances of the TIBCO iProcess Objects Director, see the *TIBCO iProcess Objects Director Administrator’s Guide*). If the *InstanceNumber* is specified, it is added to the **Key** of the SWNodeInfo object that is created. If the *InstanceNumber* is omitted, it defaults to instance number 1.

*Note - If you are manually creating an SWNodeInfo object, and you do not want a UDP broadcast to also go out, be sure to set the **IsBroadcast** property on SWEnterprise to false to turn off the default auto-discovery broadcast.*

If you use MakeNodeInfo or MakeNodeInfoEx to add an SWNodeInfo object to the NodeInfos list that is already in the list, an swLLDupErr is returned.

Example — Connecting to a Specific Node

See [page 311](#) for a comprehensive example.

```
Set oEnterprise = New SWEnterprise
  oEnterprise.IsBroadcast = False           'no broadcast
Set oNodeInfo = oEnterprise.MakeNodeInfo("seosun2", "swipe", _
                                         "10.20.30.112", 34567)
```

Configuring the TIBCO iProcess Objects Server TCP Port

The TIBCO iProcess 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 needs to be configured either **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 message to a specific node.

- **Static** - This assignment causes the TCP port number to always remain the same for that server. Use this assignment if you are manually adding a node to the NodeInfos list. This is required because you must specify the TCP port as an argument to the MakeNodeInfo / MakeNodeInfoEx method. Therefore, you must know the TCP port the server is going to be using.

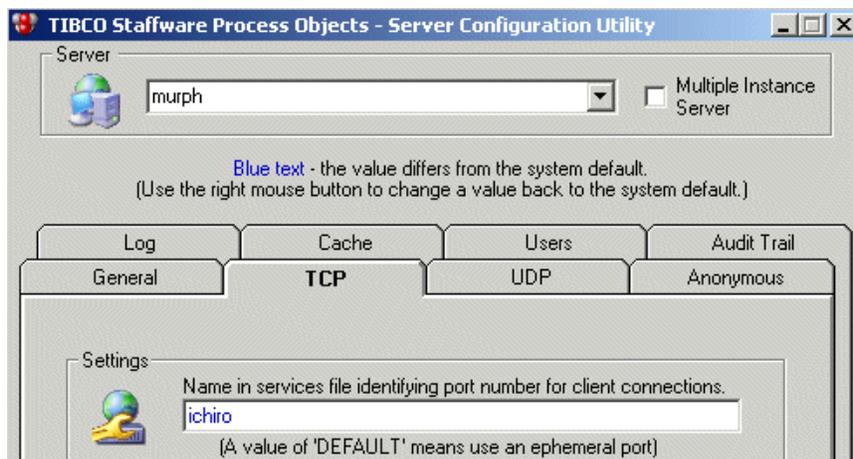
Note that all TIBCO iProcess Objects Servers that want to make use of the TIBCO iProcess Objects Director must use a static TCP port. This allows the TIBCO iProcess Objects Director to be configured with those port numbers so it knows the TCP port number to use when establishing a connection between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

Note - For information about specifying the TCP ports on which 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 and click on the **TCP** tab (as shown below).



Note - For information about using the configuration utility, see the TIBCO iProcess Objects Server Administrator's Guide.

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:

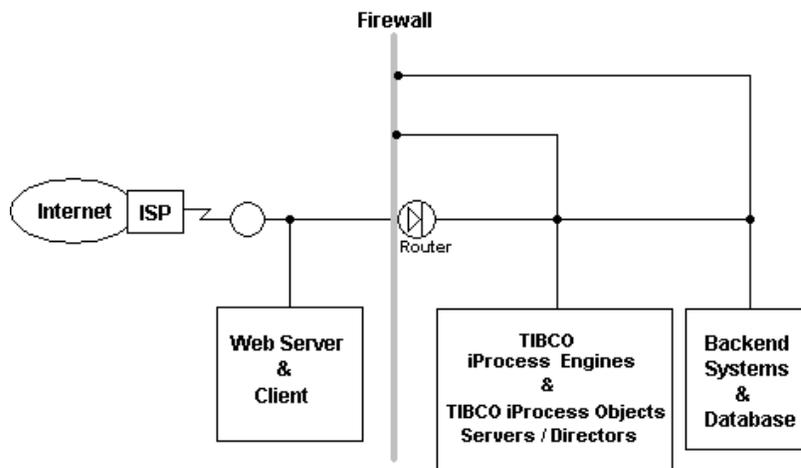
```
ichiro 6666/tcp      # TCP port assignment
```

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*.

Can I use TIBCO iProcess Objects through a Firewall?

TIBCO iProcess 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 27; 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*)
- Use the **MakeNodeInfo** or **MakeNodeInfoEx** method to manually add the TIBCO iProcess Objects Server or TIBCO iProcess Objects Director to the NodeInfos list.
- 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.



Creating Enterprise Users

The second of the two “Enterprise” objects is the **SWEntUser** object. This object contains information about a person using TIBCO iProcess Objects across the enterprise (this person may actually have different Staffware user names and/or passwords on each node). The SWEntUser object contains methods that are used to log in and out of nodes (TIBCO iProcess Objects Servers) in the enterprise. It also includes information about the nodes that the user is currently logged into in the enterprise. When a user is logged into multiple nodes across the enterprise, this object provides a single point of access to the case and work item data that the user can access on any of the logged-in nodes.

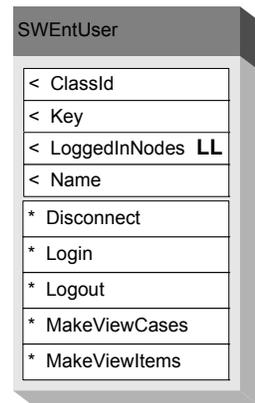
Creating an enterprise user should be done right after creating the SWEnterprise object. It is done by calling the **CreateEntUsers** method on the SWEnterprise object. This creates one or more SWEntUser objects and adds them to the **EntUsers** local list on SWEnterprise. Once the SWEntUser objects are created for the enterprise users, the **Login** method on SWEntUser can be used to log the users into one or more nodes.

This example shows the creation of three SWEntUser objects, one each for users Edgar, Jay, and Carlos (see [page 312](#) for a comprehensive example):

```
' Create a Enterprise user
  Set oEntUserAdmin = oEnterprise.CreateEntUsers("AdminUser")

' Create multiple Enterprise users with a single method call
  UserArray(0) = "Edgar"
  UserArray(1) = "User1"
  UserArray(2) = "Carlos"

oEnterprise.CreateEntUsers UserArray
```



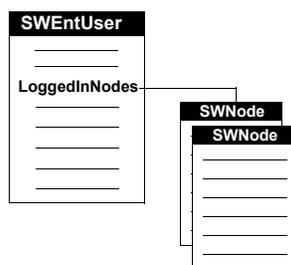
Logging In

After an `SWEntUser` object has been created for an enterprise user, the user can be logged into one or more nodes (TIBCO iProcess Objects Servers). This is done by calling the **Login** method on `SWEntUser`.

Note - For information about logging in to a TIBCO iProcess Objects Server via a TIBCO iProcess Objects Director, see the TIBCO iProcess Objects Director Administrator's Guide.

The Login method requires parameters that identify the node(s) the person is to be logged into, the person's login password, and optionally, the user name the person is known by on the node on which he is logging into. If the person is being logged into more than one node with a single Login method call, his user name and password must be the same on all of the nodes. If the person is being logged into multiple nodes, and his user name differs on each node, the Login method must be called multiple times.

In Windows systems, the password parameter provided with the Login method is authenticated against the user's O/S password in the domain in which the node is located. (See [“Turning On/Off Password Checking” on page 33](#) for information about turning off password checking.)



For each node that the user is successfully logged into, an `SWNode` object is added to the **LoggedInNodes** local list on the user's `SWEntUser` object. Each `SWNode` object contains information about work queues, work items, groups, attributes, etc., that belong to that node.

Login Example — Single Node

See [page 312](#) for a comprehensive example.

```
' Login AdminUser to "swipe" as user "swadmin"
  Set oNodeSwipe = oEntUserAdmin.Login(oNodeInfo.Key, "staffware", "swadmin")
```

Login Example — Multiple Nodes

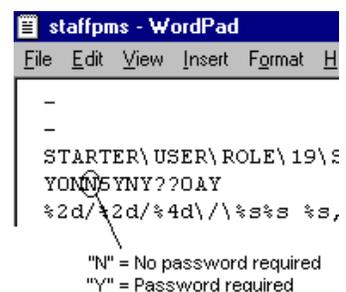
See [page 312](#) for a comprehensive example.

```
' Login AdminUser to 2 more nodes (user is swadmin, password ="staffware")
  NodeKeys(0) = oNodeInfoAIX.Key
  NodeKeys(1) = "swdoug2|doug1"
  oEntUserAdmin.Login NodeKeys, "staffware", "swadmin"
```

Turning On/Off Password Checking

Password checking can be turned on and off at two levels — at the TIBCO iProcess Engine and the TIBCO iProcess Objects Server.

- **TIBCO Process/iProcess Engine** - The `$$WDIR/etc/staffpms` (UNIX) and `SWDIR\etc\staffpms` (Windows) file can be modified to enable or disable O/S password checking. This is done by specifying a “Y” or “N” in the 4th character of line 4 in the `staffpms` file.
- **TIBCO iProcess Objects Server** - The TIBCO iProcess Objects Server `SEOPasswordRequired` configuration parameter allows you to override password checking if it’s required by the TIBCO iProcess Engine (this parameter has no affect if password checking is NOT required by the TIBCO iProcess Engine). For information about this parameter, see the *TIBCO iProcess Objects Server Administrator’s Guide*.



A property is provided on the SWNode object that allows you to determine if the user’s password has expired. The `IsUserPwdExp` property returns True if the user’s password has expired; it returns False if the user’s password has not expired. (This flag is applicable only if password checking is enabled using the methods described above.)

Logging In When Using Multiple Instances of the TIBCO iProcess Objects Server

If you are running multiple instances of the TIBCO iProcess Objects Server, the `Key` property on the `SWNodeInfo` object identifies the instance of that server. This allows you to log into a specific instance of the TIBCO iProcess Objects Server.

If you used the `MakeNodeInfoEx` method to create the `SWNodeInfo` object, the `Login` method will validate that the instance number of the server that is being logged into matches the instance number that was specified in the `MakeNodeInfoEx` method. It is possible that the server at the `TCPPort` specified in `MakeNodeInfoEx` does not have the same instance number specified in the `InstanceNumber` parameter. If the instance number of the server being logged into does not match the instance in the `NodeKey`, `swLoginFailErr` is thrown with extended text stating "server instance number mismatch".

Login Failures

Whenever there is a login error, it is *always* returned as an `swLoginFailErr` type exception. The specific details of why the login failed are in the error message string.

There is no way to return more than one error. Since you can login to more than one node with a single login, specific reasons for the login failure is written to the error message string.

You can parse the error message string (or display or log it) to see the specific errors that occurred for each specified key. You can also parse for the error-specific text (it starts with " : err ="). The example below shows the error message text that is displayed if the login failed because of a WINSOCK error:

```
SWClient Error: Logins failed on.
seosun2|i3test|N|1 : err = Winsock problem, see Client log.
```

Logging in Using a TIBCO iProcess Objects Director

Once the client has an `SWNodeInfo` object that represents a TIBCO iProcess Objects Director, the key from that object can be passed in the `NodeKeys` parameter of the **Login** method:

```
Login (NodeKeys, Password, [UserName])
```

If the `NodeKey` represents a TIBCO iProcess Objects Director (`IsDirector = Y`), the TIBCO iProcess Objects Director will use the “pick method” specified when the Director was configured to determine which TIBCO iProcess Objects Server the client should connect to. Internally, a TCP connection is established between the client and the TIBCO iProcess Objects Server that the Director selected. From the user’s standpoint, the selection and connection to the TIBCO iProcess Objects Server is transparent. All future transmissions while that user is logged in are made directly to the TIBCO iProcess Objects Server instance.

For more information about using the TIBCO iProcess Objects Director, see the *TIBCO iProcess Objects Director Administrator’s Guide*.

Logging Out

Users log out using the **Logout** method on `SWEntUser`. When a user is logged out of a node, that node’s respective `SWNode` object is removed from the **LoggedInNodes** local list on `SWEntUser`. (The `Logout` method allows you to log a user out of one or more nodes at a time.)

Calling the **Logout** method closes both the connection to the TIBCO iProcess Objects Server and the SAL session for the user. This has implications you should consider, depending on whether you are developing a thick client or a web-based client — see the “[How often should Users be Logged In and Out?](#)” section below for more information.

Logout Example — Single Node

See [page 312](#) for a comprehensive example.

```
' logout swadmin from swipe node
  oEntUserAdmin.Logout "seosun2|swipe"
```

Logout Example — All Nodes at Once

See [page 312](#) for a comprehensive example.

```
' logout swadmin from all nodes
  oEntUserAdmin.Logout
```

How often should Users be Logged In and Out?

The answer to this question depends on the type of application you are designing.

In a thick application, users will typically log in at the start of the client application and log out when the application is terminated. To minimize network traffic, and to optimize object re-use, thick applications should not be written so that the user is repeatedly logged in and out.

In a thin, web-based application, users should *not* be logged out each time they leave a web page. When a user initially logs into a node, the TIBCO iProcess Objects Server starts a SAL session for the user. Starting the SAL session, which maintains the state of the user throughout the session, is very time intensive for the TIBCO iProcess Objects Server. When the user leaves the web page, they

should be disconnected (by calling the **SWEntUser.Disconnect** method), but not logged out (i.e., don't call **SWEntUser.Logout**). The Disconnect method releases all of the user's local object references, but does not close their SAL session. When the user returns to the web page, a login is performed, but a SAL session for the user does not need to be started since there is already one open for that user. This causes all subsequent logins to be very fast. See [“Stateless Programming” on page 270](#) for more information.

When Should I use Anonymous Logins?

Anonymous logins allow a user to login as a certain Staffware user, without requiring the user to enter a password. This capability was added to support web-based applications. If the application has a need to allow unknown users to log in, anonymous logins can be used to map an anonymous user to a valid Staffware user. This means that the same Staffware user is being used by multiple people, and therefore, the application must manage any conflicts which might arise from this.

By default, anonymous logins are disabled. You can enable and configure them using the Server Configuration Utility (Windows) or the **swentobjsv.cfg** file (UNIX). The Anonymous tab from the Server Configuration Utility is shown below.

	Anonymous UserName	Staffware UserName	Inherit	Pool Size
1	Anonymous1	swanon1	yes	20
2	Anonymous2	swanon2	yes	50
3			no	
4			no	
5			no	
6			no	
7			no	
8			no	
9			no	
10			no	

This utility is used to specify anonymous usernames, the Staffware usernames they map to, whether the anonymous users inherit the admin privileges of the Staffware user, and the number of SAL sessions to pool for each anonymous user.

All access as an anonymous user is handled as if accessed by a single user. Therefore, if you want to track the person on the other end of the connection via the audit trail, you will need to use custom fields on the procedure to store their identity, or use the user-defined audit trail.

See [“Anonymous Parameters” on page 312](#) for more information about configuring the anonymous user parameters. Also see the *Stateless Programming* chapter on [page 270](#) for information about using TIBCO iProcess Objects in a web-based environment.

Database Configuration

The **SWDatabaseConfig** object exposes database configuration information on the iProcess Engine. This object can be obtained by calling the **getDatabaseConfig** method on the **SWIPEConfig** object.

Note - At the time of publication of this document, this functionality is available only in the TIBCO iProcess Objects (Java) client (not COM or C++).

The SWDatabaseConfig 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 database background user name, which defaults to "swpro".
- **getAdminPassword** - The iProcess Engine database background user's password.

SWDatabaseConfig
getProvider
getComputerName
getTCPPort
getConnectionId
getUserName
getPassword
getAdminName
getAdminPassword

Database Configuration Access

The TIBCO iProcess Objects Server contains a configuration parameter (**DBConnectionAccess**) that is used to specify whether database configuration information is available through the SWDatabaseConfig object. 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 227](#)),
- 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.

Note - At the time of publication of this document, this functionality is available only in the TIBCO iProcess Objects (Java) client (not COM or C++).

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 227](#)),
- 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

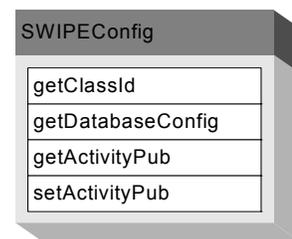
Configuration information for activity publication is stored in the database. To configure activity publication information for your iProcess Engine, you must do the following:

- 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 **SWIPEConfig** object — use of this object is described below.

Using the SWIPEConfig Object

The **SWIPEConfig** 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 SWIPEConfig 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 242](#) — 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 SURNAME LOAN_AMOUNT DEAD_REASON DEAD_DATE
Case terminated	9	\$ALL\$	ACCNO SURNAME LOAN_AMOUNT

Activity	Activity Number	Step Name	Field Name
			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>
    </MonitorList>
  </MonitorDetail>
</ProcedureMonitor>
```

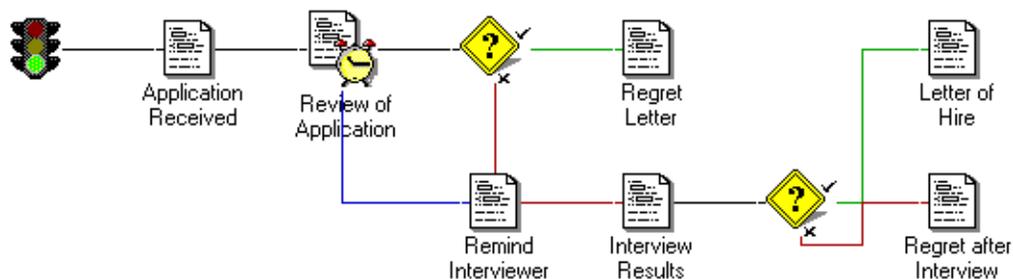
```
<Monitor>
  <ActivityList>
    <Activity Num="3"/>
  </ActivityList>
  <StepList>
    <Step Name="MTGACC01"/>
  </StepList>
  <FieldList>
    <Field Name="ACCNO"/>
    <Field Name="SURNAME"/>
    <Field Name="LOAN_AMOUNT"/>
    <Field Name="DEAD_REASON"/>
    <Field Name="DEAD_DATE"/>
  </FieldList>
</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>
```

5

Procedures

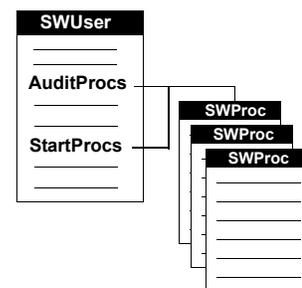
Introduction

A business process that is automated with TIBCO tools is referred to as a “procedure.” Procedures are defined with a TIBCO tool called the “TIBCO iProcess Modeler”. 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.

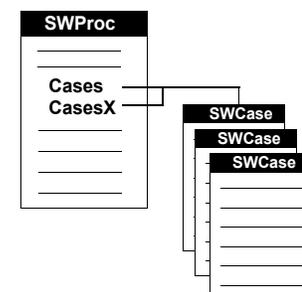


Procedures are represented by the **SWProc** object. The SWProc object contains methods that are used to start, close, and purge cases (instances of procedures), and properties that contain information about the procedure definition. You can get a list of all of the procedures that are defined on a node from the **SWNode.Procs** property

Typically, you will look at lists of procedures from the perspective of the user, specifically, the list of procedures that the user can audit or start. This is done using the **AuditProcs** and **StartProcs** properties on the **SWUser** object.



After getting an **SWProc** object, you can get either a “view” or an “XList” of all of the live cases of that procedure using the **Cases** or **CasesX** properties, respectively (views and XLists are described in “Working with Lists” on page 55). Each live case of the procedure is represented by an **SWCase** object.



See the “Case Management” chapter on page 230 for information about starting and managing cases of a procedure.

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 version i10 or newer.*)

When a procedure is created using the TIBCO iProcess Modeler, 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.

Accessing the Procedure Version Number

The major and minor version numbers are available using the following properties:

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

These properties are available on a number of objects, such as:

- **SWProc** - From this object, the version number represents the version of the procedure definition.
- **SWProcAudit** - From this object, the version number represents the version of the procedure at the time the procedure was modified.
- **SWCase** - From this object, the version number represents the version of the procedure when the case was started.
- **SWOutstandingItem**, **SWDynamicSubProcCallStep**, **SWEventStep**, **SWEAStep**, **SWGraftStep** - All of these objects represent outstanding steps. From these objects, the version number represents the version of the procedure at the time the step became outstanding.
- **SWAuditStep** - From this object, the version number represents the version of the procedure when the action represented by the SWAuditStep object was performed. 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 the TIBCO iProcess Modeler, 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.)

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 Objects, 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 Objects, 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 **SWProc.Status** property. 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 (**SWProc.Status**), 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 properties/methods act upon or list only the "current" version of a procedure (e.g., the Procs property only returns the "current" version of each procedure — in the next subsection).

Listing Versions of a Procedure

If you are operating with a TIBCO iProcess Objects Server that supports procedure versions (version i10 or newer), the **Procs** property will return a list containing the "current" version of each procedure on the node. See the previous section for information about "current" versions. (With earlier versions of the TIBCO iProcess Objects Server, the Procs property returns all procedures, regardless of their Status (except swIncomplete, which is not supported by TIBCO iProcess Objects).

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

- **SWNode.ProcGroups** - This property returns a list of **SWProcGroup** objects. Each SWProcGroup object represents a procedure defined on the node.

Then for each procedure returned by ProcGroups, you can access all of the versions available for that procedure using the following property:

- **SWProcGroup.ProcVersions** - This property returns a list of **SWProc** objects, each representing a specific version of the procedure. This allows you access to ALL versions of the procedure.

Accessing a Specific Procedure Version

The "key" for a procedure contains a procedure version component (*ProcMajorVer|ProcMinorVer*) so that you can obtain an SWProc object for a specific procedure version using the ItemByKey method.

$$\text{SWProc.Key} = \text{HostingNode|Name|ProcMajorVer|ProcMinorVer}$$

This would typically be used to extract a specific procedure version from a list of procedures returned by **SWNode.Procs**, **SWUser.AuditProcs**, **SWUser.StartProcs**, or **SWProcGroup.ProcVersions**.

Making Different Versions of Procedures

Procedures must be defined using the TIBCO iProcess Modeler. Once a procedure has been created and saved with the TIBCO iProcess Modeler, the procedure definition can be accessed using TIBCO iProcess Objects. A procedure definition is represented by an **SWProc** object.

The previous section describes the normal ways of accessing procedures that are defined on a node. These methods, however, require you to traverse the object model hierarchy to locate the desired procedure.

There are also "make" methods that allow you to "make" an SWProc object without traversing the object model hierarchy. These methods allow you to specify a version number or status in the method call:

- **MakeProc** - This method accepts optional version numbers to specify that you want the method to return a particular version of the procedure, as follows:

$$\text{MakeProc}(\text{ProcName}, [\text{NodeName}], [\text{ProcMajorVer}], [\text{ProcMinorVer}])$$

- **MakeProcByStatus** - This method allows you to specify the status of the procedure you would like returned:

$$\text{MakeProcByStatus}(\text{ProcName}, \text{ProcStatus}, [\text{NodeName}])$$

You can specify that this method return procedures that have a status of swReleased, swUnreleased, or swModel.

You can also make an SWStep object from a specific version of a procedure using the following method:

- **MakeStep** - This method allows you to optionally specify the procedure version number in the method call, as follows:

`MakeStep (StepName, [NodeName], [ProcMajorVer], [ProcMinorVer])`

Using the Tag Property to Make Specific Versions

The **Tag** property for both SWProc and SWStep contain a procedure version component, as follows:

`SWProc.Tag = NodeName|ProcName|ProcMajorVer|ProcMinorVer`

`SWStep.Tag = NodeName|ProcName|StepName|ProcMajorVer|ProcMinorVer`

This allows you to save the procedure or step tag from a specific version of procedure, then use that tag to "make" an SWProc or SWStep object for that specific version of procedure by passing the tag in the **MakeProcByTag** or **MakeStepByTag** method, respectively. Note that these methods are backward compatible with older versions of the TIBCO iProcess Objects Server that do not support procedure version control, i.e., they will accept tags with or without the procedure version component.

Procedure Version Details

The **SWProc** object also contains the following properties that provide information about the version of the procedure:

- **DateReleased** - This returns the date and time the procedure was released. If it has not been released, this property returns 12/31/3000 11:15:00 PM.
- **DateCreated** - This returns the date and time this version of the procedure was created.
- **DateModified** - This returns the date and time this version of the procedure was last modified. If this version of the procedure has not been modified, this returns 12/31/3000 11:15:00 PM.
- **DateWithdrawn** - 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.
- **LastUpdateUser** - This returns the name of the user that last updated this version of the procedure.
- **VersionComment** - This returns the comment that was entered by the user who last updated the procedure.

Procedure Audit Trails

When procedures are modified in the TIBCO iProcess Modeler, information about the modification is written to an audit trail. This procedure audit trail information is available in the following property on SWProc:

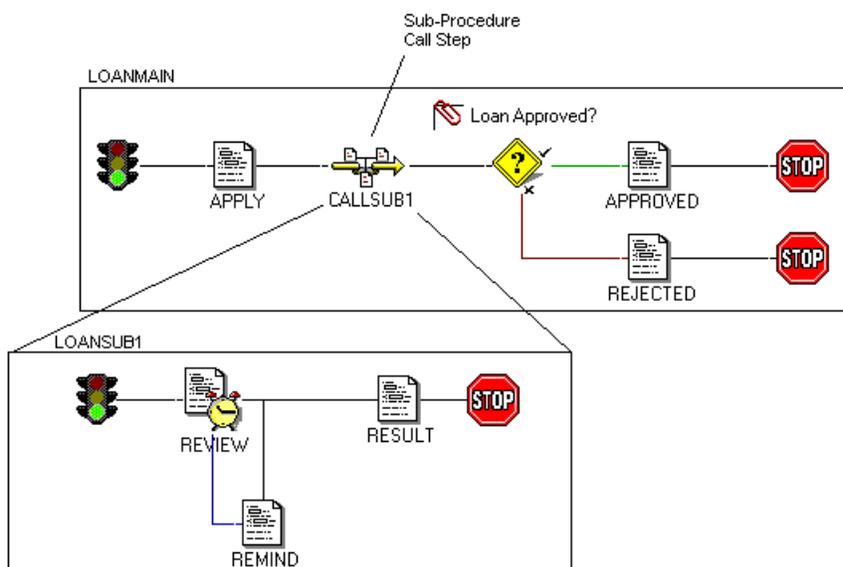
- **ProcAudits** - This property returns a list of **SWProcAudit** objects, each representing a specific modification to the procedure definition. Note that for procedure audit data to be retrieved from the server and placed in this property, you must set the **SWProc.IsWithAuditData** flag to True.

The **SWProcAudit** object contains the following properties, which provide information about the procedure modification:

- **Action** - Describes the modification made to the procedure — these actions are defined in the enumeration type **SWProcAuditActionType**.
- **Comment** - User comments concerning modifications made to the procedure.
- **Date** - The date and time the modification occurred.
- **ProcMajorVer** - The "major" portion of the procedure version number.
- **ProcMinorVer** - The "minor" portion of the procedure version number.
- **User** - 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 (as in the **CALLSUB1** step above), flow is halted along that particular path of the calling procedure until the sub-procedure has completed.

When a procedure is defined in the TIBCO iProcess Modeler, you specify that it is either a “main” procedure or a “sub-procedure”. A main procedure is started directly with the **StartCaseEx** 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 the TIBCO iProcess Modeler, 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 47](#) 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 the TIBCO iProcess Modeler, rather than specifying the sub-procedures to start, you specify the name of an array field. At run-time, the client 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 48](#) 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 259](#) 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.

The TIBCO iProcess 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 the TIBCO iProcess Modeler, and not through TIBCO iProcess Objects.

Sub-Procedure Call Steps

When a sub-procedure call step (also called a “static” sub-procedure call step) is defined in the TIBCO iProcess Modeler, you specify the name of the sub-procedure to start when the process flow reaches the sub-procedure call step. This name is available in the **SubProcName** property on the **SWStep** object that represents the sub-procedure call step (where **SWStep.Type** = **swSubProcCall**).

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 in the **SubProcStartStep** property on the **SWStep** object that represents the sub-procedure call step. If a start step other than the default start step was not specified, the **SubProcStartStep** property will return an empty string.

Sub-Procedure Start Precedence

When you start a main procedure with the **StartCaseEx** method, you can also specify a *SubProcPrecedence* parameter that allows you to specify the “precedence” of sub-procedure versions (released, unreleased, or model) that are launched from the main procedure. In other words, you are telling it which version to look for first, second, then third. For example, you can specify that it look for unreleased, then model, then released versions. The default is to start only released versions. See [“Sub-Procedure Precedence” on page 233](#) for more information.

Dynamic Sub-Procedure Call Steps

A "static" sub-procedure call step 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 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 the TIBCO iProcess Modeler, 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 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 in the **SWStep.SubProcName** property.

If no elements of the "sub-procedures to start" array field are assigned when the step is processed, the step is immediately released and its actions are performed.

See [“Array Fields” on page 94](#) for information about how array fields are used with dynamic sub-procedure calls.

- **Sub-Procedure Start Steps** - When a dynamic sub-procedure call step is defined in the TIBCO iProcess Modeler, 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 corresponding to the sub-procedures in the array field in the SubProcName property.

The name of the array field containing the names of the start steps on which to start is available in the **SWStep.SubProcStartStep** property.

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 94](#) 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 the TIBCO iProcess Modeler, you can specify an array field, whose elements will contain a return status for each corresponding sub-procedure started by the dynamic sub-procedure call step. The name of the array field containing the return statuses is reflected in the **SWStep.SubProcStatus** property. 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 shown in the table on the right.

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 in **ReturnStatus** property on the **SWSubProcStep** 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 reflected in the following properties on **SWStep**:
 - **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 — see the *TIBCO iProcess Modeler Advanced Design Guide* for information about parameter templates.)
 - **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 `SubProcName` array field are 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 the TIBCO iProcess Modeler, 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.

Passing Data between a Main and Sub-Procedure

Field values can be exchanged between a parent and child sub-case at the time a sub-case 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 the TIBCO iProcess Modeler.

The SWStep object that represents either a sub-procedure call step or a dynamic sub-procedure call step contains the following properties 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):

- **InToFldNames** - Local list of destination fields, in the sub-case, that receive values at sub-case start.
- **InFromFldNames** - Local list of source fields, from the parent case, whose values are passed to the sub-case when it starts.
- **OutToFldNames** - Local list of destination fields, in the parent case, that receive values from the sub-case when it terminates.
- **OutFromFldNames** - Local list of source fields, from the sub-case, whose values are passed to the parent case when the sub-case terminates.

Outstanding Sub-Procedures / Sub-Procedure Call Steps

Sub-procedure call steps and sub-procedures can become “outstanding” for the following reasons:

- The process flow reaches a sub-procedure call step. This causes the *sub-procedure call step* to become outstanding. The process flow is halted along that path of the procedure, and the sub-procedure is started. The sub-procedure call step remains outstanding until the sub-procedure completes, at which time the process flow will continue to the next step in the procedure.
- A sub-procedure is started by a dynamic sub-procedure call step because the process flow has reached the dynamic sub-procedure call step. This causes the *sub-procedure* that is started to become outstanding. The sub-procedure will remain outstanding until it completes.

- A sub-procedure is started by a graft step when the sub-procedure is specified in the **startGraftTask** method. This causes the *sub-procedure* that is started to become outstanding. The sub-procedure will remain outstanding until it completes.

Every outstanding sub-procedure call step and sub-procedure that is started by an outstanding dynamic sub-procedure call step or graft step results in an **SWSubProcStep** object.

It's important to note that the SWSubProcStep object can represent either a sub-procedure call step or a sub-procedure.

The **SubProcSteps** property on **SWCase** returns a list of SWSubProcStep objects, one for each *sub-procedure call step* that is currently outstanding in the case.

The **SubProcSteps** property on the **SWDynamicSubProcStep** and **SWGraftStep** objects returns a local list of SWSubProcStep objects, one for each *sub-procedure* that was started by the dynamic sub-procedure call step or graft step. This property will continue to be populated with a SWSubProcStep object for each sub-procedure that was started, even after the sub-procedure has completed. You can determine if a sub-procedure is still outstanding (hasn't completed yet) by accessing the **IsOutstanding** property on the SWSubProcStep object that represents the sub-procedure you are interested in.

SWCase.SubProcSteps
(outstanding sub-procedure call steps)

SWDynamicSubProcStep.SubProcSteps
(sub-procedures started)

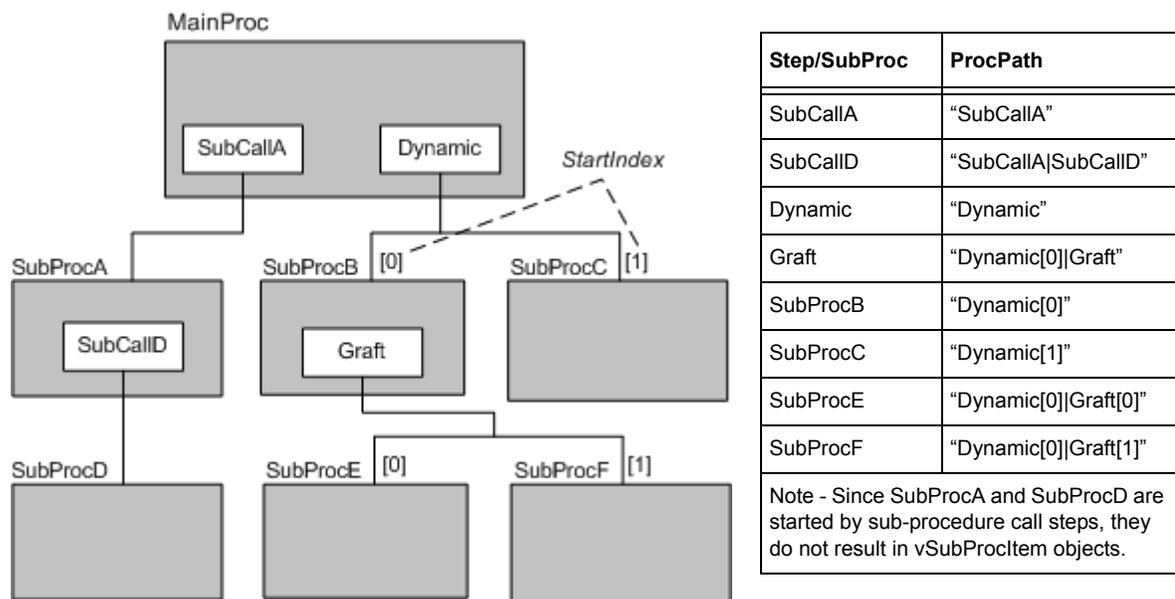
SWGraftStep.SubProcSteps
(sub-procedures started)

SWSubProcStep	
< Arrived	
< ClassId	
< Deadline	
< IsOutstanding	
< Key	
< ReturnStatus	
< StartIndex	
< StepName	
< SubCaseId	
< SubCaseNumber	
< SubCaseTag	
< SubProcMajorVer	
< SubProcMinorVer	
< SubProcName	
< SubProcNode	
< SubProcPath	

Sub-Procedure Proc Path

The **SubProcPath** property on SWSubProcStep will return either the path to a sub-procedure call step (if the SWSubProcStep object represents an outstanding sub-procedure call step) or to a sub-procedure (if the SWSubProcStep object represents a sub-procedure started by a dynamic sub-procedure call step or graft step).

The illustration below shows example strings returned by the SubProcPath property for a variety of outstanding sub-procedure call steps and sub-procedures.



If an outstanding sub-procedure call step is in the main procedure, its SubProcPath will simply consist of the name of the sub-procedure call step (see SubCallA in the example).

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

If the case family contains any dynamic sub-procedure call steps or graft steps that start multiple sub-procedures (see the Dynamic or 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. It is used in the SubProcPath to be able to identify the path through multiple sub-procedures to the desired outstanding item.

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 accessing the **StartIndex** property on the **SWSubProcStep** object that represents that sub-procedure.

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

Public Steps

When a step is defined with the TIBCO iProcess Modeler, 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 Objects do 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 the TIBCO iProcess Modeler:

Can be Public Step	Cannot be Public Step
Normal step	EIS step
Complex router step	Script step
Event step	Auto step ^a
EAI step	Open client step ^a
Sub-procedure call step	
Dynamic sub-procedure call step	
Graft step	

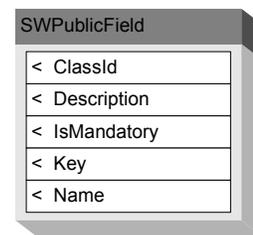
a. Not available if using a TIBCO iProcess Objects Server version i10 or newer.

The **SWStep** object has the following properties to support public steps:

- **IsPublic** - Flag that returns True if the step has been defined as a “public step”.
- **PublicFields** - For a step that has been designated as a public step, you can also specify fields as being “public fields”. The PublicFields property returns an SWList of SWPublicField objects, one for each field that has been designated as a public field on that public step.
- **PublicStepDesc** - Description of the public step (which may differ from the SWStep description).
- **PublicStepURL** - A URL that may be used as a link for additional information about the public step.

For each field on a public step that has been designated as a public field, an **SWPublicField** object is created. This object has the following read-only properties:

- **ClassId** - Identifies the class.
- **Description** - A description of the public field.
- **IsMandatory** - Flag indicating whether or not this public field is mandatory. As described above (in the PublicFields description), data entry in public fields that are flagged as mandatory is not enforced by TIBCO iProcess Objects. It is up to the application to enforce this requirement.
- **Key** - Uniquely identifies the object in a list of objects (Key = Name).
- **Name** - The name given the public field.



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

Working with Lists

Types of Lists Available

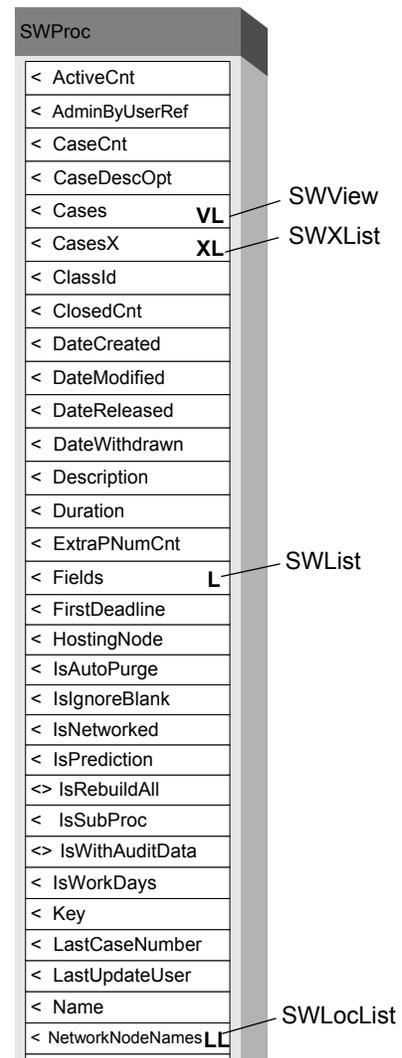
Many of the properties (methods in Java and C++) return a *list* of items when the property/method is accessed. This list is in the form of a *list object*. List objects are special objects that contain lists of other objects or strings (only SWLists and SWLocLists contain strings). There are four types of list objects:

- **SWList** - This type of list contains objects (or strings) retrieved from the TIBCO iProcess Engine, for example, all of the procedures (SWProc objects) that are defined on a node, all of the steps (SWStep objects) that are defined for a particular procedure, etc. The information in SWLists tends to be relatively static.
- **SWLocList** - These are called “local lists,” primarily because they are maintained locally by the client application. Methods are provided that can be used by the programmer to add and delete objects from the local list. Local lists can also be filled with information from the server.
- **SWView** - “Views” are a special form of list that hold only two types of objects — SWCase or SWWorkItem. The SWView object is similar to the SWList object in that it contains objects that are retrieved from the TIBCO iProcess Engine, except it also has properties that are used to filter and sort the SWCase and SWWorkItem objects. The only way a client can limit the number of objects returned from the server to the SWView is to use a filter or set the MaxCnt property.

Note - Because of improved efficiency, all new code should use SWXLists instead of SWViews.

- **SWXList** - “XLists” are very similar to views, except for one very important distinction. SWXLists are designed to allow the programmer to control the number of objects in the list at any given time and to allow non-sequential access. Therefore, only a specified number of objects from the TIBCO iProcess Engine are retrieved at one time, allowing you to minimize network traffic and maximize efficiency.

When viewing the object model graphics (provided on each distribution CD), you can determine if and what type of list is returned by a particular property or method (as shown in the illustration above).



How to Determine the Type of Object in a List

All list objects are homogeneous, containing only a single type of object or string. Every type of list object has a **Type** property that indicates the type of object in the list.

Below are enumerations that are returned in the **Type** property to identify the type of object in the list.

Objects on SWLists:

swNodeInfoID = 1	swTableID = 49
swUserID = 2	swTableFieldID = 50
swRoleID = 3	swDateValueID = 51
swProcID = 4	swTimeValueID = 52
swAttributeID = 5	swNumericValueID = 53
swGroupID = 6	swAWorkQID = 60
swWorkQID = 7	swParticipationID = 61
swBstrID = 8	swCaseDataQParamDefID = 64
swAutoFwdID = 9	swPublicFieldID = 66
swFwdItemID = 10	swOutstandingItemID = 67
swFieldID = 11	swSubProcStepID = 68
swStepID = 12	swPredictedItemID = 69
swAuditStepID = 13	swProcAuditID = 76
swOSUserID = 14	swProcGroup = 77
swFMarkingID = 43	swGraftStepID = 78
swFConditionalID = 44	swEAIStepID = 79
swFRowID = 45	swEventStepID = 80
swListValidationID = 48	swDynamicSubProcStepID = 81
	swTransControlStepID = 83

*Note the **swBstrID** enumeration shown for SWLists and SWLocLists. This is the enumeration that is returned if the list contains strings. (BSTR strings are a type of unicode string used in COM. In Java, Java strings are returned. In C++, pointers to null-terminated "C" strings are returned.)*

Objects on SWLocLists:

swBstrID = 8	swMarkingID = 23
swNodeID = 15	swEntUserID = 24
swConfigInfoID = 16	swActionID = 25
swSALInfoID = 17	swConditionalID = 26
swClientInfoID = 18	swDeadlineValueID = 27
swThreadInfoID = 19	swFMarkingID = 43
swQSessionInfoID = 20	swLabelID = 46
swActiveUserID = 21	swCaseDataQParamID = 65
swSortFieldID = 22	swCasePredictQParamID = 70
	swExtProcessID = 82

Objects on SWViews

swCaseID = 28
swWorkItemID = 29

Objects on SWXLists

swUserID = 2	swCaseID = 28
swGroupID = 6	swWorkItemID = 29
swOSUserID = 14	swPredictedItemID = 69

Note that *every* object also has a **ClassId** property that identifies that particular object. They are identified by the enumerations shown above.

Lists are Filled Asynchronously

An important thing to remember when accessing items on a list is that, by default, the items are retrieved from the server asynchronously (except SWLocLists). This means you can access the first items that have been retrieved before the download is complete.

Asynchronous transmission of data between the server and client is the most efficient means of transmission. It allows the client to get quick access to large data streams, especially when accessing a list containing thousands of cases or work items. The client can begin filling controls on the screen with data without having to wait for the server to complete sending the entire data set.

How to Force Synchronous Behavior

A property has been provided on both the SWList and SWView objects that forces synchronous behavior for item retrieval. Use this property if you want to ensure that all data has been received from the server without an error before processing it. Setting the **IsWaitForAll** property to True tells the client to not create objects from the raw data buffers and add them to the SWList or SWView until all items have been received from the TIBCO iProcess Objects Server.

This behavior comes at a price, however. None of the items in the list are available to the client until all items have been received from the server, which could be a significant amount of time with large lists.

Even though asynchronous transmission between the server and client is the most efficient, there are instances in which a programmer may want to force synchronous transmission. One reason is when data integrity is more important than speed. Applications that must process all items and want to guarantee that all data is received before processing would want to set IsWaitForAll to True. The TIBCO iProcess Objects Server only sends the status of the request in the final segment of the reply, so until the entire reply is received, the client does not know if the request completed successfully. Also, transmission errors could disrupt the reply.

In these situations, the client would have to catch the error and deal with it, even though it may have already processed some of the data received from the server. The following is an example of using the IsWaitForAll property.

Example — Synchronous Behavior

See [page 322](#) for a comprehensive example.

```
Set oWorkQList = oUser.WorkQs
oWorkQList.IsWaitForAll = True
For Each oWorkQ In oWorkQList
    Debug.Print "Name: " & oWorkQ.Name & "; Description: " _
                & oWorkQ.Description
    Debug.Print "oWorkQList.Count = " & oWorkQList.Count
Next
```

Determining the Number of Objects in a List or View

To determine the number of objects in an SWList or SWView, you need to understand how data is retrieved from the TIBCO iProcess Objects Server and added to a list. A consequence of the asynchronous transmission of data between the TIBCO iProcess Objects Server and client is that the **Count** property on SWList and SWView tells you “how many items are currently on the list at the client” (i.e., received from the TIBCO iProcess Objects Server so far).

The Count property will not match the total number of items available on the server until *all* items have been created into objects and added to the list at the client. One way to know that this has happened is to check to see if the **IsEOL** property is True. If IsEOL is True, it means “the last item you accessed is the last item on the list.”

What’s important to understand about the IsEOL property is that it’s **ONLY** True when the last item you accessed was the last item in the list. If another item in the list is subsequently accessed, IsEOL is then set to False.

In the example below, the Count property will contain the total number of items in the list because the While loop accesses all items available until it reaches the end of the list (EOL).

See [page 322](#) for a comprehensive example.

```
oWorkQList.Rebuild
  oWorkQList.IsWaitForAll = False
  i = 0
  oWorkQList.IsEOL = False
  While (oWorkQList.IsEOL = False)
    Debug.Print "Name: " & oWorkQList(i).Name & "; Description: " _
      & oWorkQList(i).Description
    Debug.Print "oWorkQList.Count = " & oWorkQList.Count
    i = i + 1
  Wend
```

Note that using **Item**, **ItemByKey**, **IsEOL**, and **Count** cause a message to be sent to the server requesting data, which always has the potential of generating an error. Because of this, you need to always include error handling code in case an error is generated as a result of the communication with the server (the comprehensive example on [page 322](#) includes this error handling code).

More information about counts and how items are accessed on lists is provided in the remaining sections of this chapter.

What about SWXLists?

Item counts work a little differently when using SWXLists in comparison to SWLists and SWViews. The SWXList object also has a **Count** property that tells you the number of items currently in the XList at the client, but it also has an **ItemCount** property that tells you the number of items that are available from the server. The ItemCount value is immediately available without having to access the items in the list because the TIBCO iProcess Objects Server creates a corresponding list that holds all of the items accessed (see “[Determining the Number of Items in an XList](#)” on [page 80](#) for more information about how ItemCount works).

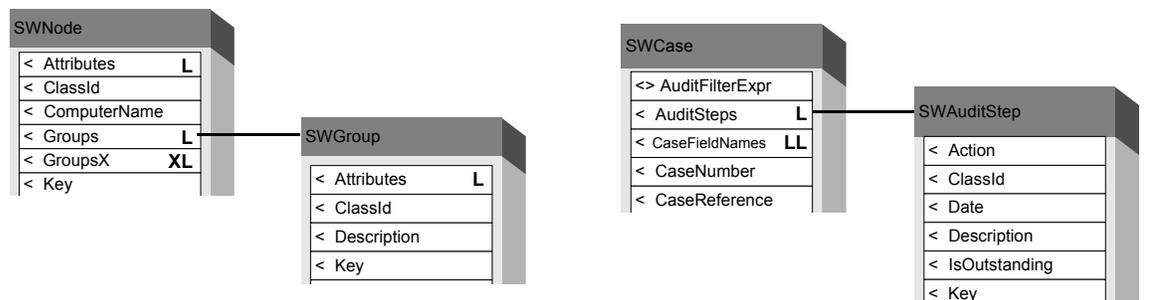
SWLists

SWLists are objects that hold lists of other objects or strings. The SWList object provides methods (Item and ItemByKey) that are used to access the items (objects or strings) that are on the SWList. It also provides properties to determine the number of items that are currently on the SWList (Count) and the type of objects it contains (Type).

The items that are in an SWList are retrieved from the TIBCO iProcess Engine. Typically, SWLists contain information that is relatively static. A couple of examples are:

- SWGroup objects in the Groups property on SWNode.
- SWAuditStep objects in the AuditSteps property on SWCase.

These examples are shown below.

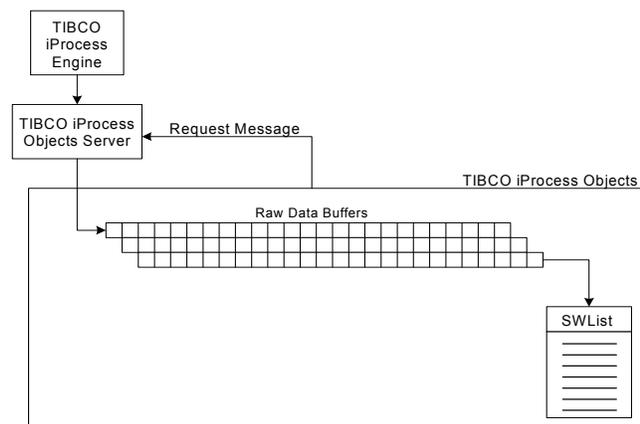


Notice the “L” next to **Groups** and **AuditSteps**, indicating that those properties return an SWList (these illustrations are from the object model graphics provided on each distribution CD).

How SWLists are Created and Populated at the Client

Creating and populating an SWList consists of these actions:

- The client requests data from the TIBCO iProcess Objects Server. This causes raw data to be sent from the TIBCO iProcess Engine to the TIBCO iProcess Objects Server, then to the raw data buffers on the client.
- Objects are created from the raw data, then added to the SWList on the client.



Although there are several properties and methods that, when accessed, cause a message to be sent to the TIBCO iProcess Objects Server requesting that data be sent to the raw data buffers on the client, objects are created and added to the SWList only when you attempt to access the objects on the SWList. This is done for efficiency reasons — if a large number of objects (some lists contain tens of thousands of objects) were immediately created and added to the SWList, a significant amount of system resources and memory may be consumed even though you only intend to access a few items on the list.

What Causes Raw Data to be Sent to the Client?

When certain properties and methods are executed, a message is sent to the TIBCO iProcess Objects Server requesting that the most current data be sent from the TIBCO iProcess Engine to the raw data buffers on the client. Which properties and methods cause the message to be sent depends, however, on whether or not the SWList is currently empty.

An SWList may be empty for the following reasons:

- It's the first time the list has been accessed since it was created.
- The **Clear** method has been called on the list.

If the SWList is currently empty, any one of these properties/methods will cause a message to be sent to the TIBCO iProcess Objects Server requesting the updated data:

- **IsEOL** - Note that in COM, this is a read/write property. When you “access” the property, i.e., you determine if its value is True or False, it sends a message to the server requesting the most recent data. Whereas, if you “set” its value to True or False, it does not send a message to the server requesting the most recent data. Likewise, in C++ and Java, `getIsEOL` sends a message requesting updated data; `setEOL` does not.

```
oProcs.IsEOL = false           `does not send a message to server
While Not (oProcs.IsEOL)      `sends a message requesting data
```

- **Count** - Even though this property sends a message to the TIBCO iProcess Objects Server requesting updated data, it does not immediately tell you how many items are in the list. This is because of the asynchronous transmission between the server and client. See [“Determining the Number of Objects in a List or View” on page 58](#) for more information.
- **Rebuild** - This method sends a message to the TIBCO iProcess Objects Server requesting updated data.
- **Item** - This method is attempting to access a specific item on the list based on an index. But the list is empty, so it sends a message to the TIBCO iProcess Objects Server requesting data, then returns the requested item to the client. For more information, see the next subsection, [“What Causes Objects to be Created and Added to an SWList?”](#).
- **ItemByKey** - This method is attempting to access a specific item on the list based on a key. But the list is empty, so it sends a message to the TIBCO iProcess Objects Server requesting data, then returns the requested item to the client. For more information, see the next subsection, [“What Causes Objects to be Created and Added to an SWList?”](#).

If the SWList currently contains items, the following method sends a message to the TIBCO iProcess Objects Server requesting the most recent data:

- **Rebuild** - This method causes the current list at the client to be cleared and a message to be sent to the TIBCO iProcess Objects Server requesting the most recent data.

What Causes Objects to be Created and Added to an SWList?

After data has been requested from the server and received in the raw data buffers on the client, it is held in the buffers until you attempt to access an item. At that time, the raw data is used to create the objects and they are added to the list at the client.

The following methods cause one or more objects to be created from the raw data and added to the SWList:

- **Item** - This method returns the object identified by the *index* parameter provided in the method call. If the object is not already on the list, it creates as many objects as necessary, in sequential order, up to the index value (which is zero based), and adds them to the SWList. For example, if you request “Item(25)”, objects 0-24 are created and added to the list, and the requested object is returned to the client (note that “Item” need not be included in your code, as in the example below, since it is the default method on all list objects).

```
i = 0
oWorkQList.IsEOL = False
While (oWorkQList.IsEOL = False)
    Debug.Print "Name: " & oWorkQList(i).Name & "; Description: " _
                & oWorkQList(i).Description
    Debug.Print "oWorkQList.Count = " & oWorkQList.Count
    i = i + 1
Wend
```

See [page 322](#) for a comprehensive example.

- **ItemByKey** - This method returns the object identified by the *key* parameter provided in the method call (see “[Object Keys](#)” on [page 84](#) for a complete list of the keys that are used with this method). If the object is not already on the list, it creates as many objects as necessary, in sequential order, until the requested object is created or the end of the list is encountered, and adds them to the SWList.

```
Set oWorkQ = oWorkQList.ItemByKey("swadmin@doug1|R")
```

See [page 322](#) for a comprehensive example.

Why do Item and ItemByKey return a Variant (COM only)?

The Item and ItemByKey methods return a variant on SWList and SWLocList because these lists can contain either objects (“IDispatch pointers”) or strings (BSTRs). Variants are the only type of variable (in COM) that can hold either a string or a pointer to an object.

Unlike SWLists and SWLocLists, SWViews and SWXLists can contain only objects (SWViews can contain SWCase or SWWorkItem objects; SWXLists can contain SWCase, SWWorkItem, SWGroup, SWUser, or SWOSUser objects). Therefore, the Item and ItemByKey methods return only IDispatch pointers when the items are on an SWView or SWXList.

When Should I Rebuild an SWList?

Typically, client applications will have no need to rebuild SWLists. Only if you are writing an administrative-type application will the need arise. This is because the data on SWLists is relatively static, so there's no need to update it.

The **Rebuild** method on SWList causes:

- the current list of items at the client to be cleared,
- the **Count** property to be set to zero,
- the **IsEOL** property to be set to False, and
- a message to be sent to the server requesting the most recent data to be sent to the raw data buffers on the client.

When the program accesses an item, data from the raw data buffers is created into objects and placed on the SWList.

How to Rebuild Subordinate Lists

For objects that are not lists but that have a Rebuild method (SWCase, SWProc, SWWorkItem, and SWWorkQ), the default behavior is to not rebuild subordinate lists that are on the object. A property is provided that, when set to True, causes all subordinate lists (this includes SWLists, SWViews, and SWXLists) on the object to also be rebuilt when you rebuild the object. This is done with the **IsRebuildAll** property. Note that setting IsRebuildAll to True causes ALL subordinate lists to be rebuilt — SWViews, SWLists, and SWXLists.

The following objects contain the **IsRebuildAll** property:

- SWCase
- SWProc
- SWWorkItem
- SWWorkQ

In the example below, when the SWProc object is rebuilt, all other lists on the SWProc object are also rebuilt:

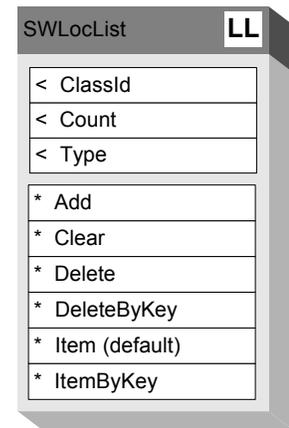
```
oProc.IsRebuildAll = True
oProc.Rebuild
```

Important - Use the IsRebuildAll property with caution. It can potentially cause very large amounts of data to be retrieved from the TIBCO iProcess Objects Server unnecessarily. It's much better to rebuild individual lists as needed.

SWLocLists

SWLocLists (local lists) are used to store lists of objects or strings that are primarily static data. The objects and strings in these lists are added to and deleted from the SWLocList in the following ways:

- By invoking the SWLocList **Add** and **Delete** methods in the client application. **CaseFieldNames** and **SortFields** are examples of local lists you might add to with the Add method.
- By invoking methods on other objects. An example of this is using the **CreateEntUsers** and **DeleteEntUsers** methods on SWEnterprise to add and delete SWEntUser objects to and from the **EntUsers** local list.
- Some local lists are filled by messages that fill other lists above them in the hierarchy.



Note that the **Count** property on SWLocList is always current since nothing is being transferred from the server for this type of list at the time the list is created.

There are only two objects that can be explicitly instantiated and added to a local list — SWSortField and SWMarking objects. All other local lists of objects are managed by other objects and cannot be directly modified by the application developer (i.e., cannot add or delete). (See “[Sorting Work Items and Cases](#)” on page 178 for an example of how SWSortField objects are added to a local list.)

How to Add Objects/Strings to Local Lists

The **Add** method on SWLocList is used to add objects or strings to a local list. Be aware of the fact that there may already be items in the local list when you add new items. For this reason, you may need to clear the list before adding the new items.

You can include an index with the Add method to specify where to insert the new item. If you do not include an index, the item is appended to the list. If the index is greater than the number of items in the list, it will also be appended to the list.

The Add method returns the index (zero based) of the item’s location in the local list. The index can be used to access the item (with the Item method), or to delete the item from the list (with the Delete method).

Example of adding a string to a local list:

```
oWorkItems.CaseFieldNames.Clear
oWorkItems.CaseFieldNames.Add "SW_CASENUM"
```

Example of adding an object to a local list:

```
Set oSortField = New SWSortField
oSortField.FieldName = "SW_CASENUM"
oSortField.IsAscending = False
oSortField.SortAsType = swNumericSort
' Add SortField to local list
oWorkItems.SortFields.Add oSortField
```

See [page 322](#) for a comprehensive example.

How to Access Objects/Strings on Local Lists

The SWLocList object contains two methods that are used to access items that are currently on a local list:

- **Item** - This method returns an item in the local list that is identified by an index value (starting at zero). You can access a specific item using an index obtained when the item was added to the local list with the Add method. Or, you can use the index to loop through the list, as in the example shown below:
- **ItemByKey** - This method returns the object identified by the *key* parameter provided in the method call (see “Object Keys” on page 84 for a complete list of the keys that are used with this method). The example below shows the use of the SWMarking key (“Name”) to find a specific SWMarking object in the local list and set their values (note that the key is case-sensitive).

```

cnt = oWorkItems.CaseFieldNames.Count
For i = 0 To cnt - 1
    Debug.Print "CaseField Name = " & oWorkItems.CaseFieldNames(i)
Next

Set oSortField = oWorkItems.SortFields.ItemByKey("SW_CASENUM")
Debug.Print "SortField Field Name = " & oSortField.FieldName

```

See [page 322](#) for a comprehensive example.

Why do Item and ItemByKey return a Variant (COM only)?

The Item and ItemByKey methods return a variant on SWList and SWLocList because these lists can contain either objects ("IDispatch pointers") or strings (BSTRs). Variants are the only type of variable (in COM) that can hold either a string or a pointer to an object.

Unlike SWLists and SWLocLists, SWViews and SWXLists can contain only objects (SWViews can contain SWCase or SWWorkItem objects; SWXLists can contain SWCase, SWWorkItem, SWGroup, SWUser, or SWOSUser objects). Therefore, the Item and ItemByKey methods return only IDispatch pointers when the items are on an SWView or SWXList.

SWViews

SWView is another type of list object. It is very similar to an SWList object, except for the following. SWView objects:

- only contain two types of objects — SWCase or SWWorkItem,
- contain properties used to filter and sort the cases and work items,
- contain properties used to request audit information on the cases in the view, and
- contain a property (MaxCnt) that limits the number of items returned from the server.

The primary purpose of an SWView object is to provide a “view” of cases and work items that are filtered and/or sorted, allowing you to limit the number of cases/work items that are displayed, and display them in a specified order.

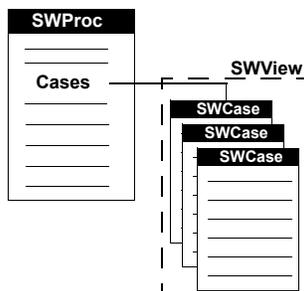
*Note - A more recent addition to TIBCO iProcess Objects is the **SWXList** object. The SWXList object provides all of the same features as the SWView, with the addition of the ability to limit the number items (cases or work items) that are retrieved from the server, making it more efficient to use. You are strongly encouraged to use SWXLists instead of SWViews. For more information, see “SWXLists” on page 73.*

Since filtering and sorting of lists of cases and work items are broad subjects themselves, and they apply to both SWViews and SWXLists, these subjects are described in-depth in their own chapters. See “[Filtering Work Items and Cases](#)” on page 152 and “[Sorting Work Items and Cases](#)” on page 178.

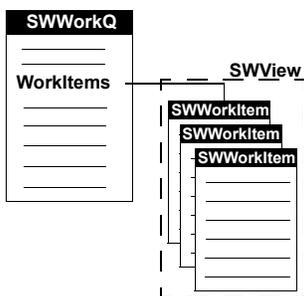
SWView		VL
<>	AuditFilterExpr	
<	CaseFieldNames	LL
<	ClassId	
<	Count	
<	ExcludeCnt	
<>	FilterExpression	
<	InvalidCnt	
<>	IsAuditAscending	
<	IsEOL	
<>	IsWaitForAll	
<>	IsWithAuditData	
<>	MaxCnt	
<	OverMaxCnt	
<	SortFields	LL
<	Status	
<	Type	
* Clear		
* Item (default)		
* ItemByKey		
* Rebuild		

The “Default” Views of Cases and Work Items

As mentioned above, views only contain cases or work items. Properties are provided that contain a “default” view of the cases on each procedure (SWProc) and the work items on each work queue (SWWorkQ). They are:



SWProc.Cases - The **Cases** property on SWProc contains a view (SWView) of all of the cases in the procedure that match the filter criteria in the **SWView.FilterExpression** property. They are listed in the view in the order specified by the **SWView.SortFields** property.



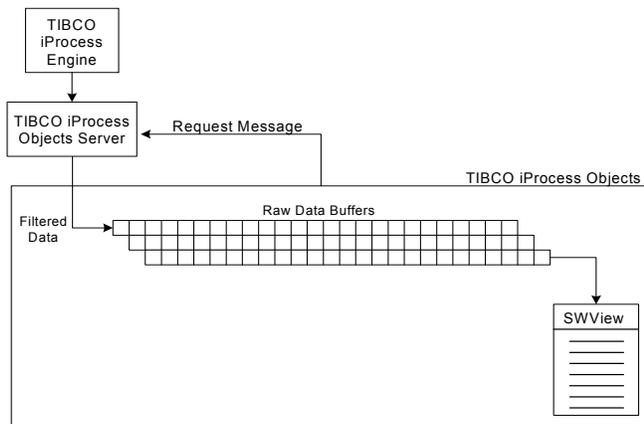
SWWorkQ.WorkItems - The **WorkItems** property on SWWorkQ contains a view (SWView) of all of the work items in the work queue that match the filter criteria in the **SWView.FilterExpression** property. They are listed in the view in the order specified by the **SWView.SortFields** property.

Like properties that contain SWLists of objects, the Cases and WorkItems properties are not automatically populated with case or work item information (i.e., SWCase or SWWorkItem objects) from the TIBCO iProcess Objects Server. The following sections describe how the case and work item information is retrieved from the server and added to the SWView.

How SWViews are Created and Populated at the Client

Creating and populating an SWView consists of the following actions:

- The client requests data from the TIBCO iProcess Objects Server. This causes raw data to be sent from the TIBCO iProcess Engine to the TIBCO iProcess Objects Server, then to the raw data buffers on the client. This raw data is filtered by the TIBCO iProcess Objects Server according to the SWView.FilterExpression property prior to being sent to the raw data buffers on the client. (See [“Filtering Work Items and Cases”](#) on page 152 for more information.)



- When objects are requested by the client application, they are created from the raw data, then added to the SWView on the client.

Although there are several properties and methods that, when accessed, cause a message to be sent to the TIBCO iProcess Objects Server requesting that data be sent to the raw data buffers on the client, objects are created and added to the SWView only when you attempt to access the objects on the view. This is done for efficiency reasons. If a large number of objects (some views contain tens of thousands of objects) were immediately created and added to the SWView, a significant amount of system resources and memory may be consumed even though you only intend to access a few of the items.

What Causes Raw Data to be Sent to the Client?

When certain properties and methods are executed, a message is sent to the TIBCO iProcess Objects Server requesting that the most current data be sent to the raw data buffers on the client. Which properties and methods cause the message to be sent depends, however, on whether or not the SWView is currently empty.

An SWView may be empty for the following reasons:

- It's the first time the list has been accessed.
- The **Clear** method has been called on the view.

If the SWView is currently empty, these properties/methods cause a message to be sent to request the most recent data on the server to be sent to the client:

- **IsEOL** - Note that in COM, this is a read/write property. When you “access” the property, i.e., you determine if its value is True or False, it sends a message to the server requesting the most recent data. Whereas, if you “set” its value to True or False, it does not send a message to the server. Likewise, in C++ and Java, getIsEOL sends a message requesting updated data; setEOL does not.

```
oWorkItems.IsEOL = False           'does not send a message to server
While Not (oWorkItems.IsEOL)      'sends a message requesting data
```

- **Count** - Even though this property sends a message to the TIBCO iProcess Objects Server requesting the most recent data to be sent to the client, it does not immediately tell you how many items are in the view. This is because of the asynchronous transfer of data between the server and client. See [“Determining the Number of Objects in a List or View” on page 58](#) for more information.
- **Rebuild** - This method sends a message to the TIBCO iProcess Objects Server requesting the most recent data to be sent to the client.
- **Item** - This method attempts to access a specific item in the view based on an index. But the list is empty, so it sends a message to the TIBCO iProcess Objects Server requesting the most recent data to be sent to the client, then returns the requested item to the client. For more information, see the next subsection, [“What Causes Objects to be Created and Added to an SWView?”](#).
- **ItemByKey** - This method attempts to access a specific item in the view based on a key. But the list is empty, so it sends a message to the TIBCO iProcess Objects Server requesting the most recent data to be sent to the client, then returns the requested item to the client. For more information, see the next subsection, [“What Causes Objects to be Created and Added to an SWView?”](#).

If the SWView currently contains items, this method causes the most recent data on the server to be sent to the client:

- **Rebuild** - This method causes the current view at the client to be cleared and a message to be sent to the TIBCO iProcess Objects Server requesting the most recent data.

Exception when Rebuilding an SWView containing Work Items

Rebuild works a little differently if the view contains SWWorkItem objects. In this case, TIBCO iProcess Objects make use of the caching that’s done by the TIBCO iProcess Engine. To determine if new work item data should be sent to the client, the following conditions are evaluated:

- If there has been a membership change in the work queue, i.e., new work items have been added or existing work items released, new work item data will be retrieved from the server (in this case, the SWView.Status property contains **swChanged**). If there has not been a membership change in the work queue, work items are NOT retrieved from the server (in this case, the SWView.Status property contains **swNoChange**).

There is an exception to this behavior; if a work item is routed back to your own work queue, the Status will always show **swNoChange**. In these situations, you can force a rebuild by first calling the **Clear** method on the SWView. This, however, should only be done after careful consideration of the performance impact created by completely repopulating the view.

- If only the status of the work items on the work queue has changed (i.e., items have been locked or unlocked), only the status information is retrieved from the server. (If there has been a status change, the SWView.Status property contains **swStatusOnly**.)

When rebuilding an SWView of work items, and the view’s status is **swStatusOnly**, the following properties on the work items in the view are updated:

- IsLocked
- IsLongLock

- LockedBy
 - IsUnopen
 - IsUrgent
 - IsDeadlineExp
- If the filter criteria (SWView.FilterExpression) or the sort criteria (SWView.SortFields) for the SWView have been modified, it is assumed that the list of work items will be different, and therefore, they will be retrieved from the server.

The reason Rebuild works this way when rebuilding views of work items is because a work queue can contain an extremely large number of work items. Retrieving all of them regardless of status causes an unnecessary amount of network traffic.

After rebuilding a view of work items, you should check the SWView.Status property to determine if you need to refresh any of your application data.

What Causes Objects to be Created and Added to an SWView?

After data has been sent from the server to the raw data buffers on the client, it is held in the buffers until you attempt to access an item (with Item or ItemByKey). At that time, the raw data is used to create the objects and they are added to the view at the client.

The following properties and methods cause one or more objects to be created from the raw data and added to the SWView:

- **Item** - This method returns the object identified by the *index* parameter provided in the method call. If the object is not already in the view, it creates as many objects as necessary, in sequential order, up to the index value (which is zero based), and adds them to the SWView. For example, if you request “Item(25)”, objects 0-24 are created and added to the view, and the requested object is returned to the client (note that “Item” need not be included in your code, as in the example below, since it is the default method on all list objects).

```

i = 0
oWorkItems.IsEOL = False

While Not (oWorkItems.IsEOL)
  Set oWorkItem = oWorkItems(i)
  Debug.Print "WorkItem Key= " & oWorkItem.key & ", CaseNum = " _
    & oWorkItem.Case.CaseNumber & ", Fields returned = " _
    & oWorkItem.Case.Fields.Count
  For Each oField In oWorkItem.Case.Fields
    Debug.Print "      FieldName = " & oField.Name & ", _
      Field Value = " & oField.Value
  Next
  i = i + 1
Wend

```

See [page 322](#) for a comprehensive example.

- **ItemByKey** - This method returns the object identified by the *key* parameter provided in the method call (see “Object Keys” on [page 84](#) for a complete list of the keys that are used with this method). If the object is not already in the view, it creates as many objects as necessary, in sequential order, until the requested object is created, and adds them to the SWView.

```
Set oWorkItem = oWorkItems.ItemByKey(key)
```

See [page 322](#) for a comprehensive example.

When should a View be Rebuilt?

Whenever the application needs access to the most current information on the server, you should rebuild the view with the **SWView.Rebuild** method. Also, after modifying the view's filter criteria (SWView.FilterExpression) or the sort criteria (SWView.SortFields), you must rebuild the view to cause a message to be sent to the TIBCO iProcess Objects Server to apply the new criteria.

The **Rebuild** method on SWView causes:

- the current view of items at the client to be cleared,
- the **Count** property to be set to zero,
- the **IsEOL** property to be set to False, and
- a message to be sent to the TIBCO iProcess Objects Server requesting the most recent data to be sent to the raw data buffers on the client.

Note - If the view contains work items, Rebuild may or may not perform the functions listed above, depending on the value of the Status property of the view. See [“What Causes Raw Data to be Sent to the Client?”](#) on page 66 for more information.

The following is an example of rebuilding a view of work items after changing the filter criteria:

```
oWorkItems.FilterExpression = "SW_PRONAME = ""TestPro4""
oWorkItems.Rebuild
Debug.Print "<=== After View Rebuild with fieldnames and sort criteria ===>"
  For Each oWorkItem In oWorkItems
    Debug.Print "WorkItem Key= " & oWorkItem.key & ", CaseNum = "_
      & oWorkItem.Case.CaseNumber & ", Fields returned = "_
      & oWorkItem.Case.Fields.Count
    For Each oField In oWorkItem.Case.Fields
      Debug.Print "      FieldName = " & oField.Name & ", Field Value = "_
      & oField.Value
    Next
  Next
```

See [page 322](#) for a comprehensive example.

How to Rebuild Subordinate Views

For objects that are not lists but that have a Rebuild method (SWCase, SWProc, SWWorkItem, and SWWorkQ), the default behavior is to not rebuild subordinate lists (this includes SWLists, SWViews, and SWXLists) that are on the object. A property is provided that, when set to True, causes all subordinate lists on the object to also be rebuilt when you rebuild the object. This is done with the **IsRebuildAll** property. Note that setting IsRebuildAll to True causes ALL subordinate lists to be rebuilt — SWViews, SWLists, and SWXLists.

The following objects contain the **IsRebuildAll** property:

- SWCase
- SWProc
- SWWorkItem
- SWWorkQ

In the example below, when the SWProc object is rebuilt, all other lists on the SWProc object are also rebuilt:

```
oWorkQ.IsRebuildAll = True
oWorkQ.Rebuild
```

Important - Use the IsRebuildAll property with caution. It can potentially cause very large amounts of data to be retrieved from the TIBCO iProcess Objects Server unnecessarily. It's much better to rebuild individual lists as needed.

When should I Create an Alternate View?

As described in [“The “Default” Views of Cases and Work Items” on page 65](#), there are default views available for cases (SWProc.Cases) and work items (SWWorkQ.WorkItems). These are the views you will typically use. However, if you have a need to show a list of work items or cases in more than one way, additional SWView objects can be created. For example, you may want to show a list of work items sorted by priority and also (concurrently) a list of unopened work items that arrived today. To do this, you would create an additional SWView containing SWWorkItem objects filtered and sorted in the desired way.

The following methods are available to create alternate views:

- **MakeViewCases** - Creates an additional SWView containing SWCase objects.
- **MakeViewItems** - Creates an additional SWView containing SWWorkItem objects.

Each of these methods are available on several objects. The object from which you invoke the method governs the scope of the method. The table below shows the scope of each method from the objects from which they can be called.

Method	Called From	Scope
MakeViewCases	SWProc	Used to view cases in the procedure.
	SWUser	Used to view cases across multiple procedures on the same node.
	SWEntUser	Used to view cases across multiple procedures on more than one node.

Method	Called From	Scope
MakeViewItems	SWorkQ	Used to view work items in the work queue.
	SWUser	Used to view work items across multiple work queues on one node.
	SWEntUser	Used to view work items across multiple work queues on more than one node.

If your alternate view spans multiple procedures or work queues, the `SWView.Status` property will always return `swChanged`.

The “single-parameter” `SWNode.MakeViewItemsByTag` method is available to create an additional view of work items by passing a single *Tag* parameter. This method would typically be used by a web-based application. Its scope is the work queues on the node from which it is called. (For more information about using “single-parameter” methods in a web-based environment, see [“Stateless Programming” on page 270](#).)

Implied Sort on Alternate Views

If you create an alternate view of cases that spans multiple procedures, there is an implied primary sort that first sorts the cases by procedure name before any user-defined sort criteria are invoked.

Likewise, if you create an alternate view of work items that spans multiple work queues, there is an implied primary sort that first sorts the work items by work queue name before any user-defined sort criteria are invoked.

Determining the Total Number of Items Available

You can determine the total number of items available *from the server* by looping through the view and checking the `SWView.IsEOL` property. When `IsEOL` returns True, it means “the last item you accessed is the last item available to the list.” At this point, the `Count` property indicates the total number of items available from the server. See the example below.

```
i = 0
oWorkQList.IsEOL = False
While (oWorkQList.IsEOL = False)
    Debug.Print "Name: " & oWorkQList(i).Name & "; Description: " _
                & oWorkQList(i).Description
    Debug.Print "oWorkQList.Count = " & oWorkQList.Count
    i = i + 1
Wend
```

It’s important to understand that the `IsEOL` property is **ONLY** True when the last item you accessed was the last item in the list. If another item in the list is subsequently accessed, `IsEOL` is then set to False.

Other count properties are also available on `SWView`:

- **ExcludeCnt** - Contains the number of work items or cases that were not included in the indexed collection at the server because they did not satisfy the criteria in the `FilterExpression` property. (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)

- **InvalidCnt** - Contains the number of work items or cases not included in the indexed collection because they were invalid within the context of the filter criteria (e.g., the filter expression references a field name not defined in all work items). (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)

How do I Limit the Number of Work Items/Cases in a View?

The **MaxCnt** property on SWView allows you to specify the maximum number of objects to retrieve from the server. This can be used to retrieve only the number of objects the client needs, for example, the number of items that will fit in a list box. The value of this property is initialized to -1, which causes all objects to be retrieved from the server.

Note that if the SWView spans multiple work queues, the MaxCnt property is applied to each queue in the view. For example, if MaxCnt is set to 10 and there are four queues in the view, 40 items will be retrieved from the server (assuming there are at least 10 items in each queue).

If MaxCnt is used, the **OverMaxCnt** property will tell you how many objects were not sent to the client because they exceeded the number specified with MaxCnt. The value of OverMaxCnt is not valid until IsEOL is True since the value wouldn't be known until all objects have been sent to the client.

How to Include Audit Data in a View

From the view level, you can specify that audit data be returned on the cases that are in the view. This is done by setting the **IsWithAuditData** property to true. If you have a list of cases in a view and you want audit data for all or most of the cases, it is much more efficient to request audit data for the entire view by setting SWView.IsWithAuditData to True. If you are interested in the audit data for a specific case, set the flag on that case only (SWCase.IsWithAuditData).

You can also specify the chronological order in which the audit data is returned with the **IsAuditAscending** property on SWView and SWCase:

- Setting IsAuditAscending to True causes audit data to be returned in ascending order.
- Setting IsAuditAscending to False causes audit data to be returned in descending order.

See “[Auditing Case Data](#)” on [page 239](#) for more information.

SWXLists

SWXLists (also called “XLists”) are similar to SWViews in that they hold a collection of objects retrieved from the server that are filtered and sorted in a particular way. The primary difference is that when an SWView is populated, it retrieves all of the objects on the server that match the filter criteria. When an SWXList is populated, however, it only retrieves a “block” of objects (specified with the **ItemsPerBlock** property), as opposed to all of the objects available. Furthermore, XLists allow the programmer to access the list randomly, unlike views where it appears to be random, but is actually sequential internally.

This approach to retrieving objects from the server allows you to more closely control the memory resources of the client, as well as lessen the amount of time the processor spends caching objects from large lists.

Note that “true” blocking only occurs with XLists containing work items — only a block of work items are retrieved at a time from the TIBCO iProcess Objects Server. When the XList contains cases, users, groups, or OSUsers, *all* items are retrieved on the first access. However, the case, user, group, or OSUser data are *not* created into objects until you access it, at which time, the objects are created a block at a time. The specifics of this are explained in the following subsections.

SWXLists work well with web-based applications. They scale well because you have control over the number of objects retrieved from the server. They also allow you to persist the lists that are created (only XLists of work items and predicted work items can be persisted) so that you can access the same collection of work items on subsequent logins. More about persistence is described in [“Working with Persisted XLists” on page 81](#).

Note - XLists have been added to improve performance when dealing with very large lists of objects. Because of the efficiencies and scalability of XLists, you should be using them instead of SWViews. They should also be used instead of SWLists that contain large numbers of groups, users, and OS users.

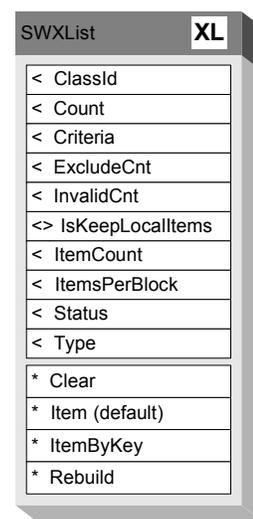
Since filtering and sorting of lists of cases and work items are broad subjects in themselves, and they apply to both SWViews and SWXLists, these subjects are described in-depth in their own chapters. See [“Filtering Work Items and Cases” on page 152](#) and [“Sorting Work Items and Cases” on page 178](#).

What types of Objects can be found in an XList?

XLists can contain the following types of objects:

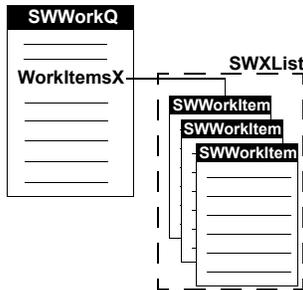
- SWWorkItem
- SWCase
- SWGroup
- SWUser
- SWOSUser

The reason these types of objects use XLists is because they are the types of objects that would most likely be in large numbers on the server — especially SWWorkItem and SWCase objects. The SWCase and SWWorkItem objects also change often on the server, requiring lists of them to be refreshed frequently.

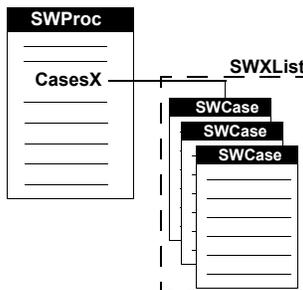


Accessing the “Default” XLists of Objects

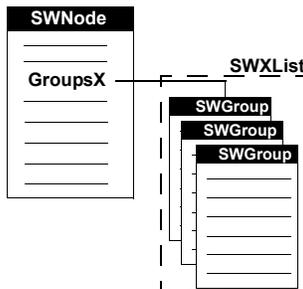
The SWWorkQ, SWProc, and SWNode objects contain properties that create “default” XLists containing the objects listed above. These properties are described below:



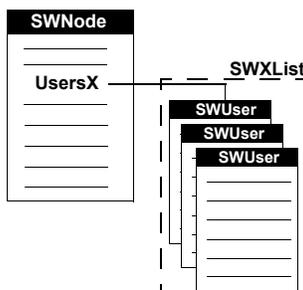
SWWorkQ.WorkItemsX - This property creates an XList containing work items on the queue that match the filter criteria specified in the SWCriteriaWI.FilterExpression property. They are listed in the XList in the order specified by the SWCriteriaWI.SortFields local list.



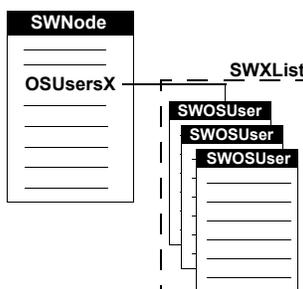
SWProc.CasesX - This property creates an XList containing cases in the procedure that match the filter criteria specified in the SWCriteriaC.FilterExpression property. They are listed in the XList in the order specified by the SWCriteriaC.SortFields local list.



SWNode.GroupsX - This property creates an XList containing the groups that are defined on the node.



SWNode.UsersX - This property creates an XList containing the users that are defined on the node.



SWNode.OSUsersX - This property creates an XList containing the OS users that are defined on the node.

Creating Additional XLists

When you want to create additional SWXList objects containing work items or cases that are filtered and/or sorted differently from the “default” XLists you get from WorkItemsX and CasesX (work items and cases are the only two types of objects you can filter and sort), use the following methods.

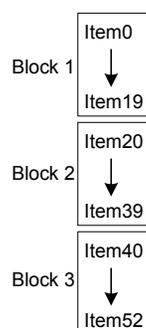
- **SWWorkQ.MakeXListItems** - This method creates an additional SWXList object containing SWWorkItem objects. This would be used in situations where the client application needs two or more views of the same work items.
- **SWWorkQ.MakeXListItemsEx** - This method creates an additional SWXList object containing SWWorkItem objects. This would be used in situations where the client application needs two or more views of the same work items. This method is an extension of MakeXListItems; it allows you to specify the specific CDQPs and case fields to return from the server when work items are accessed from the XList.

Note - You must use one of the methods above (MakeXListItems or MakeXListItemsEx) to create an XList that you are going to persist — see “Working with Persisted XLists” on page 81.)

- **SWProc.MakeXListCases** - This method creates an additional SWXList object containing SWCase objects. This would be used in situations where the client application needs two or more views of the same cases.

How XLists are Created

When you access one of the SWXList properties or call methods that create XLists (WorkItemsX, CasesX, GroupsX, UsersX, OSUsersX, MakeXListItems, and MakeXListCases), an indexed collection of the items is created on the TIBCO iProcess Objects Server. This list is filtered and/or sorted according to the filter and sort criteria, if they have been specified (See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#) and “[Sorting Work Items and Cases](#)” on [page 178](#) for information). It is also organized according to “blocks,” the size of which is dictated by the **ItemsPerBlock** property (note that this property is read-only; it can be specified only by a parameter on the Rebuild, GetXList, MakeXListItems, and MakeXListCases methods). An example is illustrated below.



In this example, the block size is 20. There are a total of 53 items in the XList.

This indexed collection of items created on the TIBCO iProcess Objects Server is NOT sent to the client until you access an item in the XList (with the Item or ItemByKey method — these are described later).

After the indexed collection (XList) has been created on the TIBCO iProcess Objects Server (with WorkItemsX, CasesX, etc.), you can refresh the list or obtain count information using the following methods:

- **Rebuild** - This method on SWXList causes a message to be sent to the TIBCO iProcess Objects Server requesting that the indexed collection on the TIBCO iProcess Objects Server be refreshed.

If the optional **ItemsPerBlock** parameter is specified (and is different than the current setting) with the Rebuild method, the XList at the client is cleared and the TIBCO iProcess Objects Server will construct a new indexed collection of items based on the new block size. A block of items, however, will not be sent to the client until the client requests an item with the Item or ItemByKey method.

If the XList contains work items, the following conditions are evaluated before determining if the XList on the TIBCO iProcess Objects Server should be rebuilt:

- If there has been a membership change in the work queue, i.e., new work items have been added or existing work items released, new work item data will be retrieved from the server (in this case, the SWXList.Status property contains **swXLChanged**). If there has not been a membership change in the work queue, work items are NOT retrieved from the server (in this case, the SWXList.Status property contains **swXLNoChange**).

There is an exception to this behavior; if a work item is routed back to your own work queue, the status of that work item will show **swNoChange**. In these situations, you can force a rebuild by passing True in the *IsRecreate* parameter on the **Rebuild** method — this forces the rebuild even though the status shows no change.

- If only the status of the work items on the work queue has changed (i.e., items have been locked or unlocked), only the status information is updated on the XList on the TIBCO iProcess Objects Server. (If there has been a status change, the SWXList.Status property contains **swXLStatusOnly**.)
- If the filter criteria (SWCriteriaWI.FilterExpression) or sort criteria (SWCriteriaWI.SortFields) for the SWXList have been modified, it is assumed that the list of work items will be different, therefore, the indexed collection of work items on the TIBCO iProcess Objects Server is rebuilt.
- **ItemCount, ExcludeCnt, InvalidCnt** - These “count” properties cause a message to be sent to the TIBCO iProcess Objects Server to obtain the current item count information. These properties do not, however, cause a block of items to be sent to the client (that is done with the Item and ItemByKey methods). These properties return the following count information:
 - **ItemCount** - This returns the total number of items in the indexed collection on the TIBCO iProcess Objects Server. If it is an XList containing cases or work items, this count includes only those cases or work items that match the filter criteria specified in the FilterExpression property.
 - **ExcludeCnt** - This property, which only applies to XLists containing cases or work items, contains the number of cases or work items that did not satisfy the Boolean expression specified in the FilterExpression property, and therefore, were not included in the XList. (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)
 - **InvalidCnt** - This property, which only applies to XLists containing cases or work items, contains the number of work items or cases not included in the indexed collection because they were invalid within the context of the filter criteria (e.g., the filter expression references a field name not defined in all work items). (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)

Controlling Resources

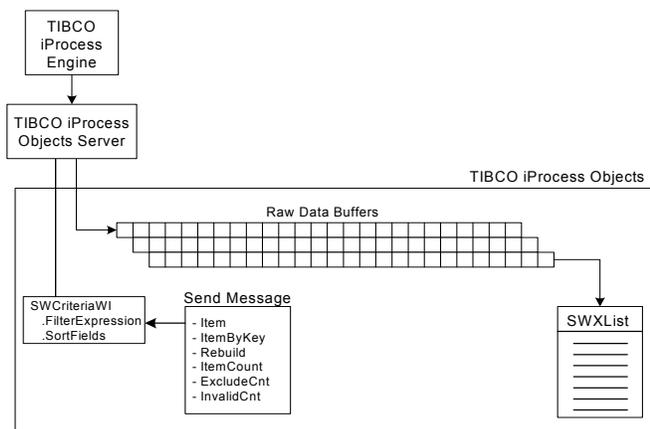
The SWXList object has an **IsKeepLocalItems** property that allows you to specify whether or not multiple blocks of items should be kept locally as subsequent blocks are retrieved from the server.

- Setting **IsKeepLocalItems** to **True** causes multiple blocks to be kept locally.
- Setting **IsKeepLocalItems** to **False** (the default) causes the previous block to be removed from the XList when a new one is retrieved from the server, i.e., only one block is kept locally at a time.

Populating an XList of Work Items

The XList on the client is populated by using the **Item** and **ItemByKey** methods. However, before determining if a message should be sent to the TIBCO iProcess Objects Server to retrieve data to populate the XList on the client, these methods look to see if the item identified by the *index / key* parameter currently exists in the XList on the client.

If the item is currently in the XList on the client, it is returned to the client application. No items are retrieved from the server.

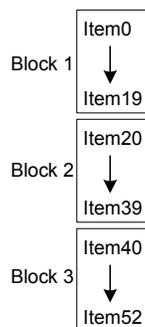


If the item is not in the XList on the client,

a message is sent to the TIBCO iProcess Objects Server to retrieve the block that contains the requested item (the size of the block is determined by the value in the **ItemsPerBlock** property on SWXList). The block is retrieved from the TIBCO iProcess Objects Server, and the objects are created and added to the XList. The requested item is then returned to the client application.

The important thing that makes accessing work items on an XList different from the other types of objects is that when a block of work items is retrieved from the TIBCO iProcess Objects Server and sent to the raw data buffers on the client, objects are immediately created from the raw data and added to the XList.

The following illustrates retrieving a block of work items from the XList on the TIBCO iProcess Objects Server. Assume a block size of 20, with 53 items in the list (the item count is zero based):



If you request item 40:

Item (40)

only Block 3 (items 40-52) is sent to the XList on the client since that is the block that contains the requested item. The requested item is returned to the client application. The Count property will equal 13.

If you then request item 50 (Item (50)), since the requested item is already in the XList on the client, no message is sent to the TIBCO iProcess Objects Server; the requested item is returned to the client application. The Count property will still equal 13.

If you then request item 25 (Item (25)), Block 2 is retrieved from the TIBCO iProcess Objects Server, since that block contains the requested item. The requested item is returned to the client application. The Count property with then equal 33.

*Note - The example above assumes you are maintaining multiple blocks of work items locally. This is controlled with the **IsKeepLocalItems** property — see “[Clearing Local Blocks of Work Items](#)” on [page 78](#) for information about this property.*

The following is an example of iterating through an XList of work items using the Item method.

```

cnt = 0
For Each oWorkItem In oWorkItemsX
    Debug.Print ("WorkItem Key= " & oWorkItem.key & ", CaseNum = " _
        & oWorkItem.Case.CaseNumber & ", Fields returned = " _
        & oWorkItem.Case.Fields.Count)
    For Each oField In oWorkItem.Case.Fields
        Debug.Print ("      FieldName = " & oField.Name & ", Field Value = " _
            & oField.Value)
    Next
    cnt = cnt + 1
    If cnt > 25 Then
        Exit For          ' only display first 25 or less
    End If
    If (cnt Mod blksize) = 0 Then ' minimize memory use on client
        Debug.Print "      XList clear "
        oWorkItemsX.Clear ' the block of 5 since already displayed info
    End If
Next

```

See [page 339](#) for a comprehensive example.

Clearing Local Blocks of Work Items

The blocks of work items retrieved from the TIBCO iProcess Objects Server and maintained in the XList consume local memory. The following are provided to control this memory usage:

- **IsKeepLocalItems** property - This read/write property on SWXList allows you to specify whether or not multiple blocks of items should be kept locally as subsequent blocks are retrieved from the server. This allows you to conserve local memory by specifying that only one block at a time be held locally. If set to True, multiple blocks will be kept locally. If set to False (the default), the previous block will be removed from the XList when a new one is retrieved from the server, i.e., only one block is kept locally at a time.
- **Clear** method - This method on SWXList frees memory and removes all items from the list on the client. Note that when called from an SWXList (this method is also available on other types of lists), it clears only the list on the client, not the buffers. Which means that if you clear an XList with the Clear method, then access an item, you are getting the same buffer data as before you cleared the list. (Note the call to **oWorkItemsX.Clear** after iterating through the block of work items in the example shown above). It's very important you do this to minimize memory usage on the client, especially if there is a large number of work items in the XList.

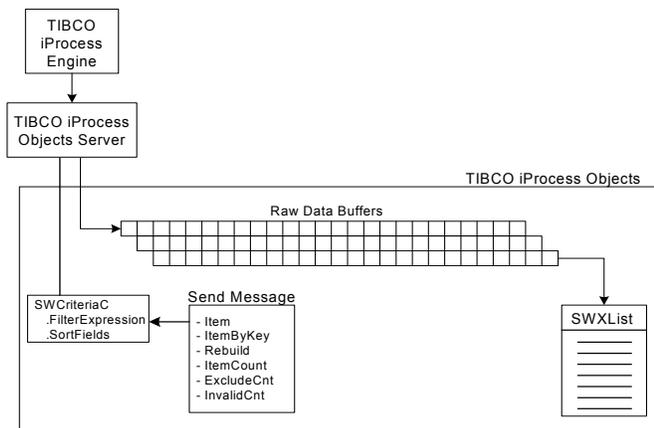
SWXLists of Work Items When Using Multiple Instances of the Server

If you are using multiple instances of the TIBCO iProcess Objects Server, be aware that an SWXList of work items is tied to a specific instance of the TIBCO iProcess Objects Server. If an SWXList of 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 an SWXList, but also to the method calls on workitems obtained from an SWXList. This is because it holds state to the Work Item Server. However, all other operations can be performed against any instance in the cluster.

Populating an XList of Cases

The first time you access any of the following properties or methods on an XList containing cases, a message is sent to the TIBCO iProcess Objects Server requesting data:

- Item
- ItemByKey
- Rebuild
- ItemCount
- ExcludeCnt
- InvalidCnt



The properties/methods listed above cause the cases in the indexed collection on the TIBCO iProcess Objects Server to be sent to the raw data buffers on the client.

The number of cases that are sent to the raw data buffers depends on the setting of the **MaxCnt** property, as follows:

- If **MaxCnt** = -1 (the default), all cases that satisfy the filter criteria are sent to the raw data buffers. So if there is a large number of cases in the procedure, the first one you access could take a significant amount of time; all subsequent accesses will be very fast.
- If **MaxCnt** is set to a value other than -1, that number of cases are sent to the raw data buffer. This allows you to limit the number of cases sent to the buffers when there is a large number of cases.

When accessing cases, they are always retrieved synchronously. That is, cases will not be added to the XList until all cases that match the filter criteria (and limited by the MaxCnt property) have been sent from the TIBCO iProcess Objects Server to the raw data buffers.

Once the data is in the raw data buffers, subsequent accesses work as follows:

- **Item** and **ItemByKey** - If the case you are requesting is currently in the XList on the client, it is returned to the client application. If the case is not currently in the XList on the client, the appropriate objects are created from the raw data buffers, then the block containing the requested case object is added to the XList. The requested case object is returned to the client application.

After iterating through a block of cases, it is very important that you call the **Clear** method on SWXList to clear the block from memory on the client. Allowing the blocks to grow on the client could adversely affect performance of your application.

- **Rebuild** - This clears the XList on the client, clears the raw data buffers, and sends a message to the TIBCO iProcess Objects Server requesting updated data, which is sent to the raw data buffers. (Note that calling the Clear method on SWXList will clear the XList on the client, but will NOT clear the raw data buffers. This is useful for minimizing memory usage, but to get a new snapshot from the server, you need to call Rebuild.)
- **ItemCount**, **ExcludeCnt**, and **InvalidCnt** - Once the raw data buffers contain data, these properties will return their respective counts. See [“Determining the Number of Items in an XList” on page 80](#) for definitions of these counts.

Populating an XList of Groups, Users, or OSUsers

Populating an XList of groups, users, or OS users works the same as for cases (as described above), except there are no filter criteria nor sort criteria with these types of objects.

After iterating through a block of groups, users, or OS users, it is very important that you call the **Clear** method on SWXList to clear the block from memory on the client. Allowing the blocks to grow on the client could adversely affect performance of your application.

Determining the Number of Items in an XList

The following counts are available for items that are stored in an XList:

- **ItemCount** - The total number of items in the XList at the server. This reflects the number of items in the XList the first time the XList was accessed or the last time it was rebuilt. If the XList contains work items or cases, this number includes only those items that satisfy the filter criteria.

This count is available immediately upon creating the XList (unlike on a view where you need to loop through the list until IsEOL is true to determine the total number of items available on the server).

```
If oWorkItemsX.ItemCount > 25 Then
    cnt = 25          ' if more than 25 only loop through 1st 25
Else
    cnt = oWorkItemsX.ItemCount ' if less than 25 loop through all of them
End If
```

- **Count** - The number of items that are currently stored in the XList at the client.
- **ExcludeCnt** - Only applies to XLists containing cases or work items. This property contains the number of cases or work items that did not satisfy the Boolean expression specified in the FilterExpression property, and therefore, were not included in the XList. (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)
- **InvalidCnt** - Only applies to XLists containing cases or work items. This property contains the number of work items or cases not included in the indexed collection because they were invalid within the context of the filter criteria (e.g., the filter expression references a field name not defined in all work items). (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)

Work Item-Specific Counts

If the SWXList contains work items, the following count properties are also available on the XList's criteria object, SWCriteriaWI:

- **DeadlineCnt** - This returns the number of work items on the XList that have deadlines.
- **UnopenedCnt** - This returns the number of work items on the XList that have not been opened.
- **UrgentCnt** - This returns the number of work items on the XList that are marked as urgent. A work item is marked as urgent if its priority value (SWWorkItem.Priority) is less than or equal to a specific value. By default, this value is 10, although it can be modified in the **staffcfg** file.

Why do Item and ItemByKey return a Variant (COM only)?

The Item and ItemByKey methods return a variant on SWList and SWLocList because these lists can contain either objects ("IDispatch pointers") or strings (BSTRs). Variants are the only type of variable (in COM) that can hold either a string or a pointer to an object.

Unlike SWLists and SWLocLists, SWViews and SWXLists can contain only objects (SWViews can contain SWCase or SWWorkItem objects; SWXLists can contain SWCase, SWWorkItem, SWGroup, SWUser, or SWOSUser objects). Therefore, the Item and ItemByKey methods return only IDispatch pointers when the items are on an SWView or SWXList.

Working with Persisted XLists

A client application can ask the TIBCO iProcess Objects Server to persist a collection of work items. (Only SWXLists containing work items and predicted work items can be persisted.) This allows the client to disconnect from the server, then reconnect at a later time and have access to the same collection of work items/predicted work items from the previous connection. The most obvious use of this feature is for web-based applications to provide access to a consistent list of work items for a given user between successive web pages.

The **SWCriteriaWI** object contains two properties that are used with persistence:

- **IsPersisted** - This property, when set to True, tells the TIBCO iProcess Objects Server to persist the XList so that it will be available at a later time.
- **PersistenceId** - This property contains a string that uniquely identifies an XList. The PersistenceId is generated by the TIBCO iProcess Objects Server and is written to this property when the XList is created. The developer is responsible for saving this ID so that it can be used later to access the persisted XList. (The PersistenceId property is also available on the SWQSessionInfo object. A queue session is started for each work queue that is opened. If the work items in the work queue are in an XList that is persisted, the PersistenceId is stored with that queue session information in the SWQSessionInfo object.)

It's very important that if you intend to persist an XList, you create the XList with the **MakeXListItems** or **MakeXListItemsEx** method for work items, or the **MakeXListPredict** method for predicted work items. This is because if you create the XList with any of these methods, you will have the only pointer to the XList, and can freely destruct it. You must destruct the XList object to cause the XList to actually be persisted on the TIBCO iProcess Objects Server. When it is destructed, if IsPersisted is False, the TIBCO iProcess Objects Server throws the list away; if IsPersisted is True, the TIBCO iProcess Objects Server preserves the list so it can be accessed again with **GetXList** (for work items) or **GetXListPredict** (for predicted work items).

```
Set oWorkItemsX = oWorkQ.MakeXListItems(10) ' will return 10 items per block
```

To persist the XList, set `IsPersisted` to `True`, then destruct the XList object:

```
oWorkItemsX.Criteria.IsPersisted = True           ' msg sent to server to keep list
XListId = oWorkItemsX.Criteria.PersistenceId
Debug.Print "SWXList Persisted Id = " & XListId
Set oWorkItemsX = Nothing                       ' release XList
```

A persisted XList is re-created by using the **GetXList** method (for work items) or **GetXListPredict** (for predicted work items). These methods require a *PersistenceID* parameter, which you must have previously saved for an XList that you wanted the TIBCO iProcess Objects Server to preserve. `GetXList` and `GetXListPredict` also have an optional *ItemsPerBlock* parameter that allows you to specify the number of items to download from the server to the persisted XList.

```
Set oWorkItemsX = oNode.GetXList(XListId)
```

Invoking the `GetXList` and `GetXListPredict` methods also clears the `IsPersisted` property, therefore, if you want the XList to remain persisted, you must set `IsPersisted` to `True` before the XList object is destructed.

The Developer is responsible for ensuring that a persisted XList is freed when you are done with it by setting the `IsPersisted` property to `False` and then destructing the SWXList object. As noted earlier, however, setting the `IsPersisted` property to `False` is done automatically if you retrieve the persisted XList with the `GetXList` or `GetXListPredict` method.

```
oXList.Criteria.IsPersisted = false             'free the xlist
.
.
set oXList = Nothing
```

See [page 339](#) for a comprehensive example.

*Note - Invoking the **SWEntUser.Logout** method causes all persisted XLists for that user to be lost. Instead of logging out, use the **Disconnect** method to disconnect from the TIBCO iProcess Objects Server. This logs the user out, but causes the user's SAL session to remain open. When the user returns, use the **Login** method to re-login the user to the TIBCO iProcess Objects Server. Any persisted XLists for that user will still be available.*

There is also a TIBCO iProcess Objects Server configuration parameter called **WQSAbandonedPeriod** that specifies how long the TIBCO iProcess Objects Server will maintain a persisted XList that has not had any activity before it will discard it. The default is 900 seconds. See [page 322](#) in the “[Tuning the SPO Server](#)” chapter for more information about this parameter.

Using Multiple Instances of the Server or Director

SWXLists of work items or predicted work items are tied to a particular instance of the TIBCO iProcess Objects Server. Therefore, if you are using multiple instances of the TIBCO iProcess Objects Server, or you used a TIBCO iProcess Objects Director to connect to the TIBCO iProcess Objects Server, you must ensure you access XLists of work items or predicted work items on the same instance of the TIBCO iProcess Objects Server on which they were created.

How to Include Audit Data in an XList

From the XList level, you can specify that audit data be returned on the cases that are in the XList. This is done by setting the **IsWithAuditData** property to True. If you have a list of cases in a XList and you want audit data for all or most of the cases, it is much more efficient to request audit data for the entire XList by setting **SWCriteriaC.IsWithAuditData** to True. If you are interested in the audit data for a specific case, set the flag on that case only (**SWCase.IsWithAuditData**).

You can also specify the chronological order in which the audit data is returned with the **IsAuditAscending** property on **SWCriteriaC** and **SWCase**:

- Setting **IsAuditAscending** to True causes audit data to be returned in ascending order.
- Setting **IsAuditAscending** to False causes audit data to be returned in descending order.

See [“Auditing Case Data” on page 239](#) for more information.

CIList (Java Only)

This object is the "Common Interface for List Objects." At this time, this list object is applicable only to TIBCO iProcess Objects (Java).

Every class implementing this interface is guaranteed to have this set of methods, and an instance of that class can be assigned to variables of the interface's type.

SWCIList eliminates the need for type knowledge of the object being used as long as you know the object implements the common interface. The SWList, SWLocList, and SWView objects all implement SWCIList. Therefore, you can assign any of these list objects to an SWCIList variable and manipulate it, without having to worry about whether it is a list, local list, or view.

SWCIList
clear
count
getClassId
getType
isEOL
isWaitForAll
item
itemByKey
items
rebuild
setEOL
setWaitForAll

Using an SWCIList

There may be methods specific to one of the list classes that you want to use that are not implemented in the SWCIList interface. In this instance, the object will have to be assigned to a variable of the specified list type in order to access the list-specific methods (e.g., the add method on SWLocList, or the getSortFields method on SWView).

The actual classes implementing the interface may implement the methods in unique or inconsistent ways. For example, the **isWaitForAll** and **rebuild** methods of the SWCIList interface in an SWLocList object are not meaningful in the context of a local list. Local lists are created for local use and have no server message directly associated with them. Therefore, **isWaitForAll** and **rebuild** have no effect in the local list. If you call the **clear** method on a list or view, then reference an item, the underlying class will send a message and repopulate the list; this is not so with local lists. If you clear a local list, there is no way to automatically repopulate the list. The user of the SWCIList interface should be aware of the behavior of the underlying objects.

Object Keys

Each of the list objects has a **Key** property that contains a string that identifies the item in the list. The key can be used to identify the item when using the **ItemByKey** method. The key for each object is shown in the table below. The objects that are not listed in this table do not have keys.

Note - Keys are case sensitive.

Object	Key
SWAction	StepName Type
SWActiveUser	UserName
SWAttribute	Name
SWAuditStep	Name (ListIndex: 1 to n)
SWAutoFwd	HostingNode ProcName StepName
SWAWorkQ	Name@HostingNode
SWCase	ComputerName ProcNode CaseReference
SWCaseDataQParam	FieldName
SWCaseDataQParamDef	FieldName
SWCasePredictQParam	FieldName
SWClientInfo	Name
SWConditional	StepName Condition
SWConfigInfo	Name
SWDatabaseConfig	Name
SWDeadlineValue	(Type in text: Minutes, Hours, Days, Weeks, Months, Years, Date, Time)
SWDurationValue	(Type in text: Microseconds, Seconds, Minutes, Hours, Days, Weeks, Months, Years)
SWDynamicSubProcStep	ProcName StepName CaseNumber
SWEAIStep	ProcName StepName CaseNumber
SWEntUser	Name
SWEventStep	ProcName StepName CaseNumber
SWExtProcess	ExtProcessName
SWFConditional	IfRow
SWField	Name
SWFMarking	Name Row Column
SWFRow	RowNumber
SWFwdItem	QName@HostingNode (Mode: R=Release)
SWGraftStep	ProcName StepName

Object	Key
SWGGroup	Name
SWIPEConfig	Name
SWLabel	Row Column
SWListValidation	Name
SWMarking	Name
SWNode	ComputerName NodeName IsDirector InstanceNumber
SWNodeInfo	ComputerName NodeName IsDirector InstanceNumber
SWOSUser	Name
SWOutstandingItem	WorkQName MailID
SWParticipation	"Participation"<Index + 1>
SWPredictedItem	ProcName StepName AddrToName
SWProc	HostingNode Name ProcMajorVer ProcMinorVer
SWProcAudit	Integer (ListIndex: 1 to n)
SWProcGroup	HostingNode Name
SWPublicField	Name
SWQSessionInfo	ClientName
SWRole	Name
SWSALInfo	UserName
SWSortField	FieldName
SWStep	Name
SWSubProcStep	StepName SubCaseNumber
SWTable	Name
SWTableField	Name
SWThreadInfo	ID
SWTransControlStep	ProcName StepName CaseNumber
SWUser	Name
SWWorkItem	AddrToName MailID
SWWorkQ	Name@!HostingNode Mode (Mode: T=Test or R=Release)

Fields & Markings

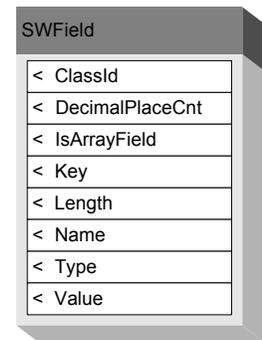
What is a Staffware Field?

A Staffware field (SWField object) represents a field that is defined in a TIBCO procedure. Before a field can be placed on a form, the field must be defined in the procedure using the TIBCO iProcess Modeler.

Note - The term “Staffware field” is a remnant of the original software created by Staffware, which was purchased by TIBCO. This term is still used in the TIBCO iProcess Engine and TIBCO iProcess Modeler dialogs and documentation, and will be used here until the engine and modeler dialogs are changed.

SWField objects are available in two properties:

- **SWProc.Fields** - This property returns an SWList of SWField objects, one for each field that is defined in the procedure. These SWField objects will never have a value in the **Value** property (it's empty) because they are only field definitions — they do not contain any data.
- **SWCase.Fields** - When accessed from an SWCase object, the Fields property returns an SWList of SWField objects, one for each field that has been returned in the live case. These SWField objects contain data in the **Value** property. (Note that case data is not available through the Value property until *after the work item has been released*. If the case has just been started, then “kept” in the starting user’s work queue, there will not be any data in the Value property because the work item has not been released yet — see [“Case Data vs. Work Item Data” on page 91](#) for more information.)



SWField	
<	ClassId
<	DecimalPlaceCnt
<	IsArrayField
<	Key
<	Length
<	Name
<	Type
<	Value

To provide the user with more control over resource usage, field data is not returned unless it is requested. For SWField objects to be returned in the Fields property on the case, you must specify which fields you want returned by doing one of the following:

- Using the **MakeWorkItemEx** or **MakeWorkItemByTagEx** methods. The *CaseFieldNames* argument on these methods allows you to specify which case fields to return from the server when the work item is created. The case fields specified are returned by the Fields property of the SWCase object (which is returned by the SWWorkItem.Case property). If the *CaseFieldNames* argument is omitted or an empty string is passed, no case fields are returned.
- Using the **MakeXListItemsEx** method. The *CaseFieldNames* argument on this method allows you to specify which case fields to return from the server when an SWWorkItem in the Xlist is accessed. The case fields specified are returned by the Fields property of the SWCase object (which is returned by the SWWorkItem.Case property). If the *CaseFieldNames* argument is omitted or an empty string is passed, no case fields are returned.

- Using the **CaseFieldNames** property. Adding field names to the CaseFieldNames property causes those fields to be returned in the Fields property when the case is returned from the server. By default, the CaseFieldNames list is empty, which means by default no fields are returned. If you want the case's fields (with data) to be returned from the TIBCO iProcess Objects Server to the case on the client, you can explicitly state which fields to return by listing them in the CaseFieldNames local list. Since CaseFieldNames is a local list, you use the **Add** method to add a field name to it — see below:

```
Set oCaseList = oProc.Cases
oCaseList.CaseFieldNames.Clear
oCaseList.CaseFieldNames.Add lboFields.Text

' Rebuild the list to pick up the requested fields
oCaseList.Rebuild
```

You can also add the special “**&ALL&**” value to CaseFieldNames to cause all fields that are defined in the procedure to be returned in the Fields property.

```
oWorkQ.WorkItems.CaseFieldNames.Clear
wString = "&ALL&"
oWorkQ.WorkItems.CaseFieldNames.Add wString
oWorkQ.WorkItems.Rebuild
```

Use **&ALL&** with caution, however, as it can result in a significant amount of data being returned.

Since cases can be accessed in a number of ways (individually, on a view, or on an XList), the CaseFieldNames property is available from a number of objects:

- SWCase
- SWView
- SWCriteriaWI
- SWCriteriaC

If the cases for which you need data are in a view or XList, but you only need data on a few of the cases, it is better to use the CaseFieldNames property on the individual SWCase objects.

If case data is needed for most of the cases in a view or XList, it is much more efficient to specify the list of field names on the SWView or SWXList objects (for an XList, you actually specify the CaseFieldNames on the XList's criteria objects, SWCriteriaC and SWCriteriaWI). This causes the TIBCO iProcess Objects Server to return the fields for every case in the view or XList in a single request to the server.

If you modify the CaseFieldNames property on an individual SWCase object, you must rebuild the SWCase.Fields list to retrieve the field data from the TIBCO iProcess Objects Server. (Or you could optionally set the IsRebuildAll property to True on the case, then rebuild the SWCase object. This is what you should do if you are also getting audit data, so it's all done in one trip to the server, making it more efficient.)

If you modify the CaseFieldNames property on a SWView or SWXList object, you must rebuild the SWView or SWXList to retrieve the updated Fields list from the TIBCO iProcess Objects Server.

What are Markings?

A “marking” is a field that is associated with a specific step in a procedure. Placing the field on a Staffware form makes the association.

Note - The term “Staffware form” is a remnant of the original software created by Staffware, which was purchased by TIBCO. This term is still used in the TIBCO iProcess Engine and TIBCO iProcess Modeler dialogs and documentation, and will be used here until the engine and modeler dialogs are changed.

Markings are represented by two objects:

- **SWFMarking** - This object represents the marking as it is defined on a Staffware form at design time. It has properties that define the physical characteristics of the marking on the form, but it does not have a Value property, i.e., it does not contain data.

SWMarking	
< ClassId	
< IsArrayField	
< IsChanged	
<> IsSendValue	
< Key	
<> Name	
<> Value	
<> ValueType	

SWFMarking	
< ClassId	
< Column	
< DecimalPlaceCnt	
< ExprValidations	LL
< Font	
< Help	
< IsArrayField	
< Key	
< Length	
< ListNames	LL
< Name	
< Row	
< Type	
< ValueType	

Each Staffware form (SWForm object) contains a list (in the FMarkings property) of the markings that are defined on that particular form, with one SWFMarking object in the list for each marking defined. (A field may exist on a form more than once, resulting in more than one SWFMarking object with the same name.)

- **SWMarking** - This object represents the marking as it appears at runtime, in the context of a live case. It has a Value property that contains the data in the marking.

SWMarking objects associated with a live case are available in the **Markings** property on SWWorkItem. However, the Markings list is NOT populated until the work item is locked (with the LockItem, LockItems, LockItemsEx, LockItemMarkings, or LockItemsMarkings methods). (SWMarking objects are also available in the Markings property on SWStep, but since these markings are not associated with a live case, they do not return any data, although they have a Value property.) The markings that are added to the Markings property are controlled differently, depending on which method is used to lock the work item, as follows:

- **LockItem** – The *allMarkings* parameter allows you to specify that either all markings on the form (both visible and conditional), or only the visible markings, be returned.
- **LockItems** – All markings on the form of each work item are returned. This method is used only if the work items are on an SWView.
- **LockItemsEx** – All markings on the form of each work item are returned. This method is used only if the work items are on an SWXList.
- **LockItemMarkings** – The *MarkingNames* parameter is used to specify the specific markings that are to be returned, allowing you to control resource usage.
- **LockItemsMarkings** – The *MarkingNames* parameter is used to specify the specific markings that are to be returned, allowing you to control resource usage. This method is used only if the work items are on an SWXList.

Note - It's possible to lock a work item, but not have any markings in the Markings list. This can occur if the work item was locked using TIBCO iProcess Objects, then the work item was destroyed (e.g., the client crashed). To reacquire the markings, you must relock the work item.

As data is entered into the fields (markings) on a form, this data is saved in the **Value** property of the respective SWMarking object.

Whenever the data in a marking is changed (that is, the Value property is modified), there are a couple of Boolean properties on SWMarking that are set:

- **IsChanged** - This indicates whether or not the value of the marking has changed *since it was initially delivered to the work queue*. If Value is modified, then the work item is “kept”, this property is still True on subsequent locks.
- **IsSendValue** - This flag specifies whether or not the data in the Value property should be sent to the server. It is automatically set to True whenever Value is modified. When the work item is either “kept” or “released,” if IsSendValue = True, the data is sent to the server; if IsSendValue = False, it is not sent. Since the flag is automatically set to True whenever Value is modified, you must explicitly set it to False if you don't want the data sent to the server.

The Markings property is cleared when you do any of the following:

- **Keep** the work item (KeepItem, KeepItems, or KeepItemsEx)
- **Release** the work item (ReleaseItem, ReleaseItems, or ReleaseItemsEx)
- **Unlock** the work item (UnlockItem, UnlockItems, or UnlockItemsEx)
- **Undo** changes to the work item (UndoItem, UndoItems, or UndoItemsEx)

Data that is sent to the server when you keep or release work items is written to Work Item Data and Case Data, respectively (see “[Case Data vs. Work Item Data](#)” on page 91), then the Markings list is cleared. To reestablish the Markings list, you must lock the work item again.

Type Validation on Fields/Markings

There are a couple of different *type* properties associated with fields and markings:

- **SWField.Type**
- **SWMarking.ValueType**

Both of these properties refer to the type of data that can be placed in the marking. These types are enumerated by the **SWFieldType** definition.

SWFieldType	
swText	= 'A'
swDate	= 'D'
swTime	= 'T'
swNumeric	= 'R'
swAttachment	= 'X'
swMemo	= 'F'
swCompositeTable	= 'C'
swComma	= 'N'
swTimeStamp	= 'O'
swUndefined	= 'U'

*Note - The SWFMarking object also has a **Type** property. However, unlike SWField.Type, it identifies the data-entry requirements for the marking on the form (optional, required, etc.). These are enumerated in the **SWMarkingType** definition.*

When the **Value** property of an SWMarking is set, the type of the variable passed is validated against the type specified in the SWMarking.ValueType property. 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 Value/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.

Note - User-created markings are validated at a different time. See “[Type Validation on User-Created Markings](#)” below.

User-Created Markings

Sometimes a user might want to change a field data value, even though that field wasn't included on a step's form. This can be done using "user-created markings." To create a user-created marking:

1. Instantiate an SWMarking object (the SWMarking object is one of only four objects that can be directly created — all others are internally instantiated).
2. Specify the name of the marking (SWMarking.Name). The name of the marking must match that of an existing field in the procedure.
3. Specify the type of data in the marking (SWMarking.ValueType).
4. Set the marking's value (SWMarking.Value).
5. Add the SWMarking object to the Markings local list (SWWorkItem.Markings).

```
Dim oMarking As SWMarking
Set oMarking = New SWMarking
oMarking.Name = "ACCOUNT_NUMBER"
oMarking.ValueType = swNumeric
oMarking.Value = Val("12345678")
oLocalItem.Markings.Add oMarking
```

The user-created marking will be sent to the server when the work item is kept or released.

Some caveats to remember about user-created markings:

- If the Markings list is recreated (by locking the work item again), your user-created markings will be lost; you must explicitly add them again. The Markings list will only contain SWMarking objects for the fields/markings that are defined on the Staffware form.
- If you set data on a user-created marking, then keep the work item (which clears the Markings list), there is no way to determine that value again until the work item is released (at which time the value is available in SWCase.Fields(i).Value).

Type Validation on User-Created Markings

As described earlier in this chapter, system-created markings are validated when the SWMarking.Value property is set. On user-created markings, however, the Value property can be set prior to establishing the ValueType. Because of this, user-created markings have their Value validated against their ValueType when the following methods are executed:

- KeepItem
- KeepItems
- ReleaseItem
- ReleaseItems
- TriggerEvent

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 Value/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.

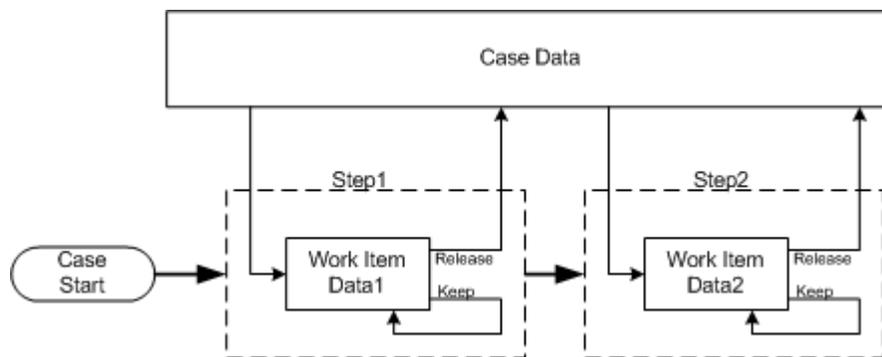
Case Data vs. Work Item Data

Data that is associated with a live case is actually of two different types — these are referred to as **Case Data** and **Work Item Data**.

Case Data is the “official” data for the case. This data, which is updated only when a work item is released, is stored in the database (Oracle, SQL, or Classic). You can also modify Case Data with the **SetCaseData** method — see “[Setting Case Data](#)” on page 92. (Case data is sometimes referred to as “central case data”.) (Note - The setCaseData method is available only if you are using a TIBCO iProcess Engine.)

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. This data reflects “keeps”, i.e., changes made to the field values in work items that are not released. When the work item is released, the data is written to the Case Data. Work Item data, which is also known as “pack data” or “packfile” data, is stored in the `SWDIR\queues\username\nodename.n` (Windows) or `$SWDIR/queues/username/nodename.n` (UNIX) directory.

Note - If you are using a TIBCO iProcess Engine, all data is stored in database tables rather than in flat files. For information about these tables, see the TIBCO iProcess Engine Administrator’s Guide.



As the flow moves to the next step, another copy of the Case Data is made to create Work Item Data for that work item.

Where do I find Case Data and Work Item Data?

Generally speaking, Case Data is found in the **SWCase.Fields** property and Work Item Data is found in the **SWWorkItem.Markings** property. Some things to note:

- **Case Data** - The Fields property on the SWCase object contains the Case Data. This can be accessed in a couple of ways:
 - `SWWorkQ.WorkItems(i).Case.Fields`
 - `SWProc.Cases(i).Fields`

The Fields property on SWProc contains a list of all of the fields defined in the procedure, but these field definitions do not contain any data. (Remember that for fields to be returned in the Fields property, you must request them through the CaseFieldNames property (as described earlier in this chapter).)

- **Work Item Data** - The Markings property on the SWWorkItem object contains the Work Item Data. This is accessed in the following way:
 - SWWorkQ.WorkItems(i).Markings

The Markings property on SWStep contains a list of the markings that are defined on the step/form, but these marking definitions do not contain any data. (Remember that the Markings property is not populated until the work item is locked.)

Setting Case Data

You can modify the Case Data for one or more fields in the case using the **SetCaseData** method on SWCase. This method requires you to pass the names of the fields and the values you want assigned to those fields:

```
SetCaseData(FieldNames, FieldValues)
```

Note - You must be using a TIBCO iProcess Engine to use the SetCaseData method.

Keeping/Releasing the Start Step

If you start a case “with field data” (see [“Starting a Case with Field Data” on page 232](#)), you also have the option of “keeping” or “releasing” the step when the case is started. 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 StartCaseEx *Release* parameter = False), the data that is passed with StartCaseEx 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 StartCaseEx *Release* parameter = True), the data that is passed with StartCaseEx 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.

Parallel Steps

Because each step has its own Work Item Data, this can create problems if your procedure has parallel steps that contain a common field. As each step is released, it copies its 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. In the end, the value of the common field in the Case Data will be the value from the last parallel step to be released.

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. See the *TIBCO iProcess Engine Administrator's Guide* for information about using **swutil**.

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 117](#), [page 144](#), and [page 169](#).

For information about using CDQP fields for sorting, see “[Using Case Data Queue Parameter Fields](#)” on [page 184](#).

Accessing Memo Fields

Fields of type **swMemo** can be accessed through the **Value** property of the **SWField** and **SWMarking** objects (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 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. To configure this “message wait time,” you must add a Registry key (Windows) or environment variable (UNIX), and set it to the number of milliseconds you would like the client to wait before timing out.

Registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Staffware plc\Staffware SEO Client\MessageWaitTime
```

Environment variable:

```
MessageWaitTime
```

If the number of milliseconds specified by **MessageWaitTime** is exceeded, the client will generate an **swTimeoutErr** error. If **MessageWaitTime** is set to 0 (zero), the client will not timeout. By default (i.e., if you do not set **MessageWaitTime**), Windows clients timeout in 30 seconds; UNIX clients timeout in 60 seconds.

For additional information about **MessageWaitTime**, see “[Message Wait Time](#)” on [page 290](#).

Accessing Attachments

You cannot directly access the data in fields of type **swAttachment**. The data in this type of field is the system-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 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 client through properties on SWWorkItem and SWCase. For example, **SW_CASENUM** is available to the client in the **SWCase.CaseNumber** property. The TIBCO iProcess Engine, however, doesn't have access to those properties, so the property names from SWWorkItem and SWCase can't be used in filter and sort expressions — instead, the system field names need to be used in your expressions. For example:

```
oWorkQ.WorkItems.FilterExpression = "SW_CASENUM=5"
```

See the *System Fields used in Filtering* tables on [page 113](#), [page 140](#), and [page 165](#) (depending on the filtering enhancements in your TIBCO iProcess Objects Server) and “[System Fields used in Sorting](#)” on [page 182](#) for lists of the system fields that can be used in filtering and sorting, respectively.

Array Fields

Array fields are defined using the TIBCO iProcess Modeler's Field Definition dialog in the same way as standard single-instance 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. For more information about indexes, see “[Array Field Indexes](#)” on [page 95](#).

Note - The terms “Staffware field and Staffware form” are remnants of the original software created by Staffware, which was purchased by TIBCO. These terms are still used in the TIBCO iProcess Engine and TIBCO iProcess Modeler dialogs and documentation, and will be used here until the engine and modeler dialogs are changed.

Array fields can be used in the same way as single-instance fields, i.e., they can be used on Staffware 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 property:

- **IsArrayField** - This Boolean flag, available on SWField, SWFMarking, and SWMarking, 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 client 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:

```
SPROCS[0] := "SUB1"
SPROCS[1] := "SUB5"
SPROCS[2] := "SUB7"
```

Note that the array field elements do NOT have to have contiguous indexes.

- **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:

```
STARTSTP[0] := "STEP1"
STARTSTP[1] := "STEP2"
STARTSTP[2] := "STEP5"
```

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 the TIBCO

iProcess Modeler. 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 Staffware 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 useful 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 Staffware 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 if your TIBCO iProcess Objects Server has CR 14434 implemented.

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."

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
```

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 8
Only the WIS Work Item Filtering enhancement (CR 12744)	Chapter 9
Both the WIS Work Item Filtering and the Database Case Filtering enhancements (CRs 12744 and 13182)	Chapter 10

Introduction

You can *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:

```
oWorkQ.WorkItems.FilterExpression = "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.

To filter work items or cases, you must set the **FilterExpression** property equal to a filter expression string (as shown in the example above). The filter expression string is evaluated against each work item in the work queue or each case in the procedure, returning either True or False. If it returns True, the work item/case is included in the view/XList; if it returns False, the work item/case is not included in the view/XList.

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. Note, however, that using some of these elements may impact how efficiently your filter expressions are processed. Details are explained 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.).

How Filtering Differs Between Views and XLists

As described in the *Working With Lists* chapter, you may be using either **SWView** objects or **SWXList** objects to hold your work item and case objects. (Remember, all new development should make use of **SWXLists** because of their improved efficiency.) The way in which filter expressions are defined is somewhat different between these two objects. The differences are described below.

Defining Filter Expressions on SWView

The **SWView** object contains a **FilterExpression** property that is used to specify a Boolean expression that defines which cases (**SWCase** objects) or work items (**SWWorkItem** objects) for the respective procedure or work queue will be returned from the server and included in the view.

Each **SWCase** or **SWWorkItem** object is evaluated against the string expression specified in the **FilterExpression** property. Those that evaluate True are included in the view; those that evaluate False are not included in the view.

After setting or modifying the **FilterExpression** property, the **Rebuild** method must be called to update the view list based on the most recent filter criteria.

Number of Work Items or Cases in the Filtered View

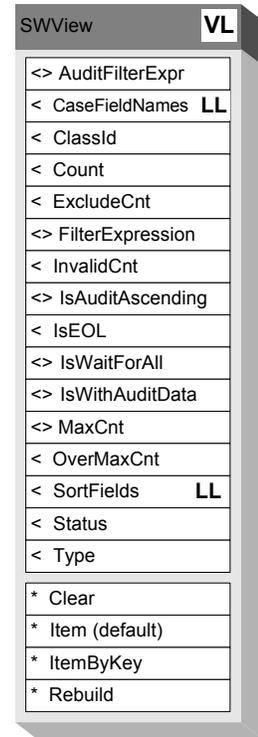
The **SWView** object has a number of properties available that provide information about the number of objects in the view:

- **Count** - This property tells you the number of work items or cases currently in the view *at the client*. You can use this property to determine the total number of work items or cases available from the server. To do this you must iterate through all of the items in the view until **IsEOL** is True. See [“Determining the Total Number of Items Available” on page 71](#) for more information and an example.

Also see [“How SWViews are Created and Populated at the Client” on page 66](#) for information about how views are populated at the client.

- **ExcludeCnt** - This property contains the number of work items or cases that did not satisfy the Boolean expression specified in the **FilterExpression** property, and therefore, were not included in the view.
- **InvalidCnt** - This property contains the number of work items or cases not included in the view because they were invalid within the context of the filter criteria (e.g., the filter expression references a field name not defined in all work items or cases).

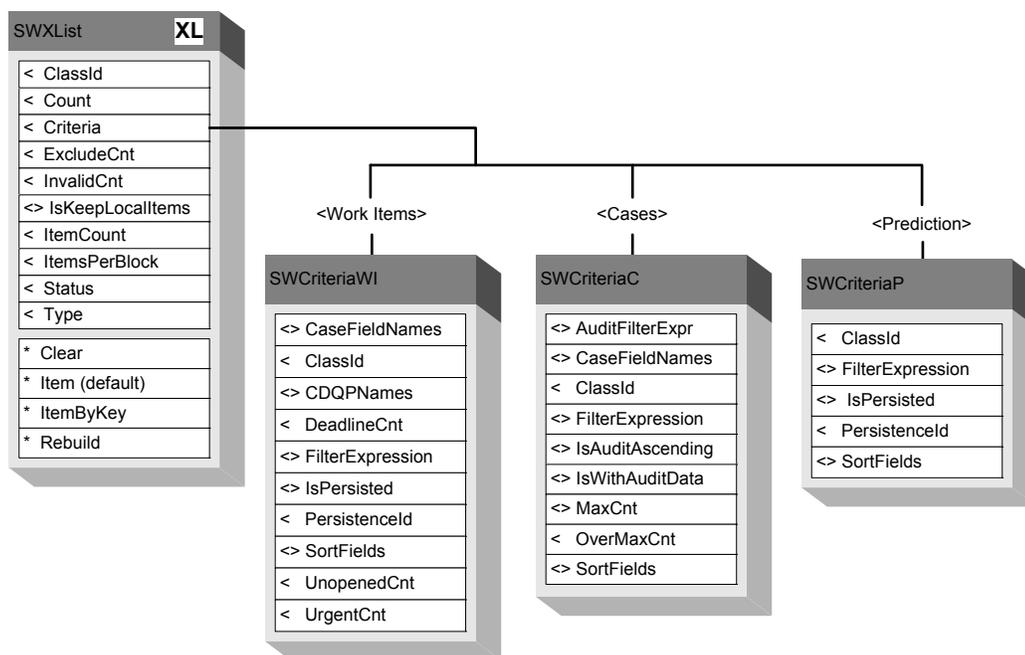
*Note - The **SWView** object also contains an **AuditFilterExpr** property that is specific to filtering **SWAuditStep** objects that are in the **AuditSteps** list of the cases that are on the view. This filtering mechanism uses its own syntax and filtering criteria; it does not use the filter criteria defined in this chapter. See [“Filtering Audit Data” on page 245](#) for information about syntax and filter criteria specific to filtering audit steps.*



Defining Filter Expressions on SWXList

The SWXList object contains a **Criteria** property that points to an SWCriteriaWI, SWCriteriaC, or SWCriteriaP object, depending whether the XList contains cases, work items, or predicted work items:

- **SWCriteriaWI** - Contains properties that specify criteria for an XList that contains **SWWorkItem** objects.
- **SWCriteriaC** - Contains properties that specify criteria for an XList that contains **SWCase** objects.
- **SWCriteriaP** - Contains properties that specify criteria for an XList that contains **SWPredictedItem** objects. (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 152](#) for more information.)



The following properties are available on the objects shown above for use when filtering cases and work items on the XList:

- **Criteria** - This property contains a reference to the appropriate criteria object (SWCriteriaWI, SWCriteriaC, or SWCriteriaP), depending on whether the XList contains work items, cases, or predicted work items.
- **FilterExpression** - This read/write property is used to specify a Boolean expression that defines which work items (**SWWorkItem** objects), cases (**SWCase** objects), or predicted work items (**SWPredictedItem** objects) for the respective work queue or procedure will be returned from the server and included in the XList.

Each SWWorkItem, SWCase, or SWPredictedItem object is evaluated against the string expression specified in the FilterExpression property. Those that evaluate True are included in the XList; those that evaluate False are not included in the XList.

After setting or modifying the FilterExpression property, the **Rebuild** method must be called to update the XList list based on the most recent filter criteria.

Number of Work Items or Cases in the Filtered XList

The SWXList object has a number of properties available that provide information about the number of objects in the XList:

- **ItemCount** - This property contains the total number of work items or cases that satisfied the filter expression and are in the XList *at the server*. This count is available immediately after the XList is created (unlike on a view where you must iterate through the objects to determine the total number available).
- **Count** - This property tells you the number of work items or cases currently available in the XList *at the client*.

See [“How XLists are Created” on page 75](#) for information about how XLists are populated with objects on the client.

- **ExcludeCnt** - This property contains the number of work items or cases that did not satisfy the Boolean expression specified in the FilterExpression property, and therefore, were not included in the XList.
- **InvalidCnt** - This property contains the number of work items or cases not included in the XList because they were invalid within the context of the filter criteria (e.g., the filter expression references a field name not defined in all work items or cases).

*Note - The SWCriteriaC object also contains an **AuditFilterExpr** property that is specific to filtering **SWAuditStep** objects that are in the **AuditSteps** list of the cases that are on the XList. This filtering mechanism uses its own syntax and filtering criteria; it does not use the filter criteria defined in this chapter. See [“Filtering Audit Data” on page 245](#) for information about syntax and filter criteria specific to filtering audit steps.*

Filtering/Sorting in an Efficient Manner

The way in which you write your filter expressions can have an effect 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.

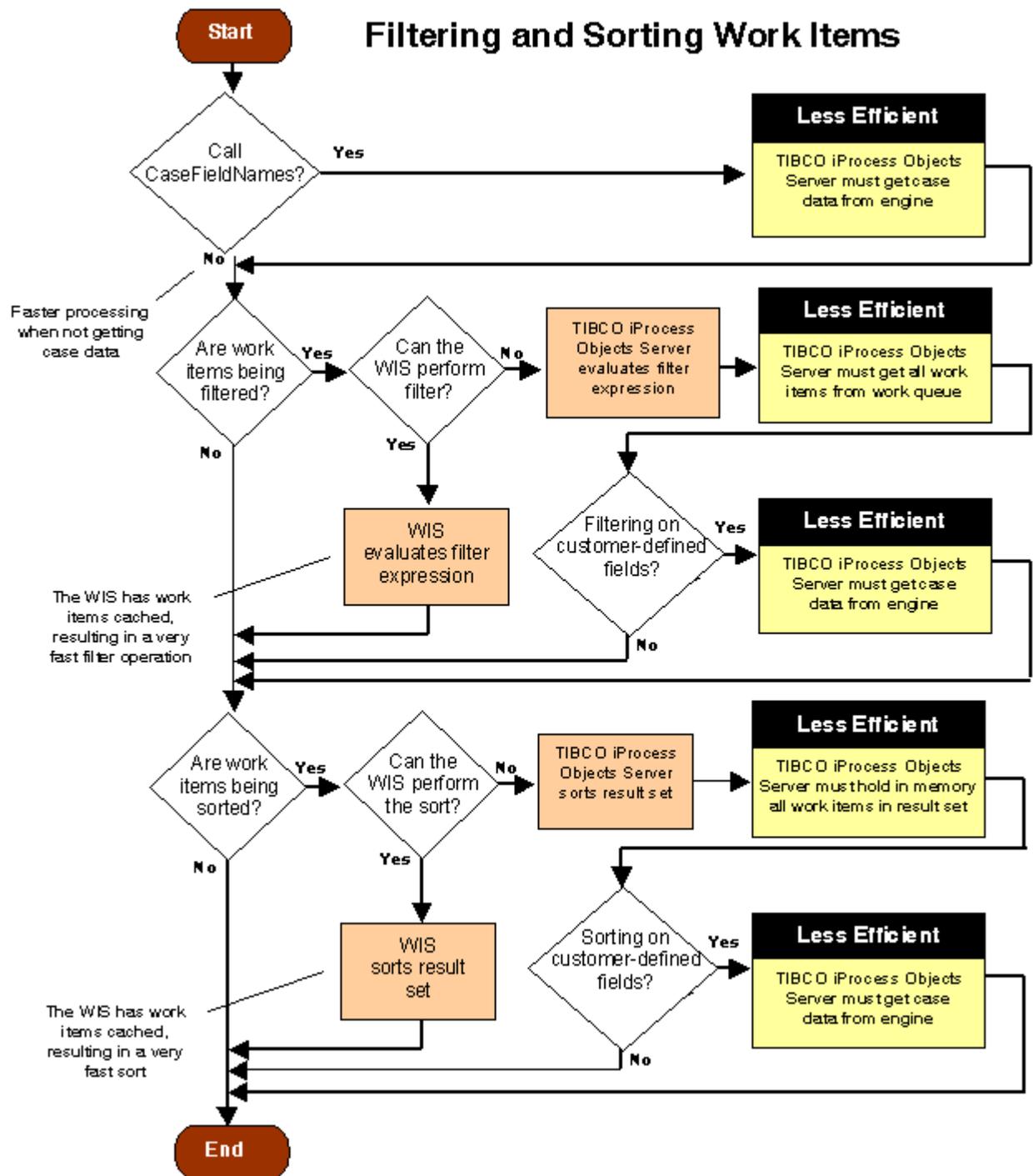
Flow diagrams (one for work items; one for cases) are shown in the following subsections that illustrate the decision process that takes place during a filter/sort operation. Note that the flow diagrams show filtering and sorting taking place in a single operation; that is the way filtering and sorting 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 105](#) 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 109](#) 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 107](#) 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:

- Calling **CaseFieldNames** - Adding field names to the **CaseFieldNames** property explicitly causes case data fields to be returned in the **Fields** property. 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 “[What is a Staffware Field?](#)” on page 86 for information about the use of **CaseFieldNames**.
- 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

Filter Criteria the WIS can Evaluate	
Element	Description
System Fields	System fields that are “WIS-compatible” (see the <i>WIS-compatible</i> columns in the table of system fields used for filtering — page 113) (They must also be applicable to filtering work items — see the <i>Applies To</i> column.)
Case Data Fields	Case data fields that have been defined as CDQPs. See “ Filtering on Case Data Fields ” on page 117 for information.
Wild Cards ^a	The wild card 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 122 for information.

- a. If the entire expression is WIS-compatible, the ‘*’ and ‘?’ are both interpreted as wild cards, 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 120 .)
Logical Operator	OR
System Fields	System fields that are NOT “WIS-compatible” (see the <i>WIS-compatible</i> columns in the table of system fields used for filtering — page 113) (They must also be applicable to filtering work items — see the <i>Applies To</i> column.)
Case Data Fields	Case data fields that have NOT been defined as CDQPs. See “ Filtering on Case Data Fields ” on page 117 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 120 .

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:

```
oCriteriaWI.FilterExpression = "SW_PRIORITY = 50"
```

The following example filter expression must be evaluated by the TIBCO iProcess Objects Server because it contains the ‘>’ comparison operator:

```
oCriteriaWI.FilterExpression = "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
<ul style="list-style-type: none"> • System fields that are “WIS-compatible”. See the WIS-compatible column in the table of <i>System Fields used in Sorting</i> on page 182. (The system fields must be applicable to filtering work items.) • Case Data Queue Parameter (CDQP) fields. See “Sorting on Case Data Fields” on page 184 for more information.
Sort Criteria the TIBCO iProcess Objects Server must Process
<ul style="list-style-type: none"> • System fields that are NOT “WIS-compatible”. See the WIS-compatible column in the table of <i>System Fields used in Sorting</i> on page 182. (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 184 for more information.

See the chapter, “[Sorting Work Items and Cases](#)” on [page 178](#) for information about setting up sort criteria.

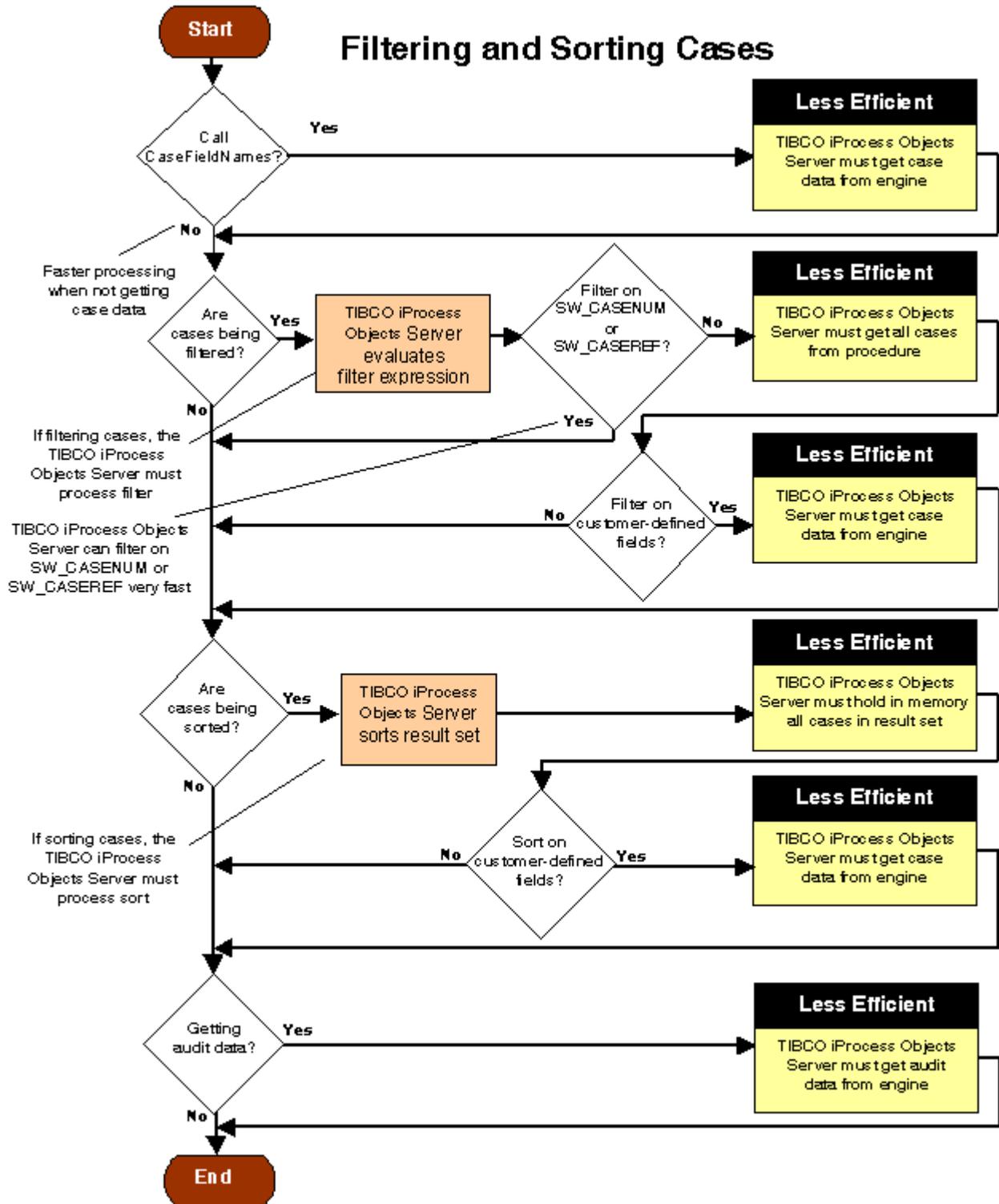
Filtering/Sorting Cases

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 110](#) 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:

- Calling **CaseFieldNames** - Adding field names to the **CaseFieldNames** property explicitly causes case data fields to be returned in the **Fields** property, which requires that the data be retrieved from the engine. See “[What is a Staffware Field?](#)” on page 86 for information about the use of **CaseFieldNames**.
- 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 and sorted *only* by the TIBCO iProcess Objects Server. This limits your options to perform an efficient filter and sort 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 120.)

Element	Description
System Fields	All system fields that are applicable to cases (see the <i>Applies To</i> column in the table of system fields used for filtering — page 113)
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.
Wild Cards	Note that the '*' and '?' characters are NOT interpreted as wild card 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 120 .

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):

```
oCriteriaC.FilterExpression = "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 examples of filtering on the case number and case reference number:

```
oProc.Cases.FilterExpression = "SW_CASENUM = 150"
```

```
oProc.Cases.FilterExpression = "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
<ul style="list-style-type: none"> • All system fields that are applicable to cases (see the <i>Applies To</i> column in the table of system fields used for sorting — page 182). • 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 the chapter, “[Sorting Work Items and Cases](#)” on [page 178](#) for specific information about setting up sort criteria.

Getting Audit Data

Getting audit data by setting the **IsWithAuditData** flag to True on the view or XList that holds your cases causes the TIBCO iProcess Objects Server to retrieve the audit data from the engine. This impacts the performance of a case filter operation.

Only include audit data in the cases in which it is needed. If you need it in all or most of the cases in the view/XList, set **IsWithAuditData** on the view/XLists. If it is needed on only one or a few cases, request it on those specific cases by setting the **IsWithAuditData** flag only on those cases.

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 113](#) 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 TIBCO iProcess Engine through the system fields is also available to the client through properties on SWWorkItem and SWCase. For example, **SW_CASENUM** is available to the client in the **SWCase.CaseNumber** property. The TIBCO iProcess Engine, however, doesn't have access to those properties, so the property names from SWWorkItem and SWCase can't be used in filter and sort criteria — instead, the system field names need to be used in your expressions. For example:

```
oWorkQ.WorkItems.FilterExpression = "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 105](#) 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 - not expired)	SW_EXPIRED	Numeric	1	X	X	

Filtering Work Items and Cases - Without Filtering Enhancements

Filter Criteria	System Field	Data Type	Length	WIS-compatible	Applies to work items	Applies to cases
Deadline set flag (1 - has deadline; 0 - does not have deadline)	SW_HASDEADLINE	Numeric	1	X	X	
Deadline time	SW_DEADLINETIME	Time	5		X	
Forwardable work item flag (1 - forwardable; 0 - not forwardable)	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 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	7	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
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 - have been open)	SW_NEW	Numeric	1	X	X	
Urgent flag (1- urgent; 0 - not 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. 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.

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 97). 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 97). 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 Staffware 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.

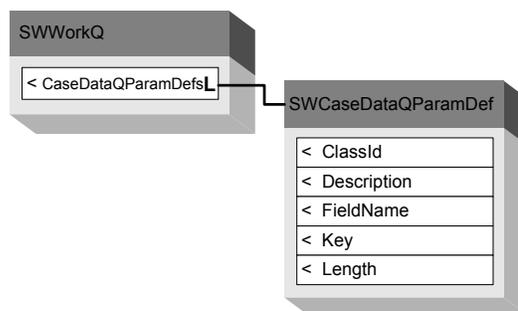
More about CDQP and work queue parameter fields are described in the following subsections.

Using Case Data Queue Parameter Fields

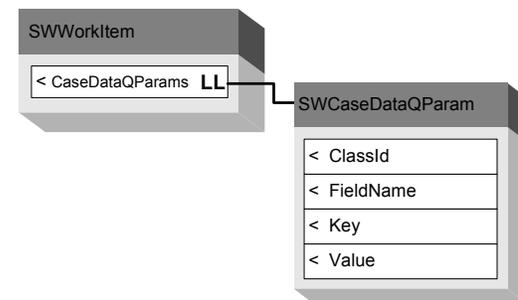
Case Data Queue Parameter (CDQP) fields provide an efficient method of filtering on the value of fields in your work items. To make use of this functionality, you must first pre-designate the fields you want to filter on as CDQP fields. Fields are designated as CDQP fields with the utility, **swutil**. 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. See the *TIBCO iProcess Engine Administrator's Guide* for information about using **swutil**.

*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 via the **SWWorkQ.CaseDataQParamDefs** property. This property contains a list of **SWCaseDataQParamDef** objects, one for each case data field that has been designated as a CDQP field in the work queue. This list tells you the CDQPs that are available for filter and sort criteria.



The **CaseDataQParams** property on **SWWorkItem** provides access to CDQPs that are being used in the work item. Note, however, to control resource usage, you can specify which CDQPs to return from the server by using the **CDQPNames** property on the **SWCriteriaWI** object. By default, all CDQPs used in a work item are returned from the server. The **SWCaseDataQParam** object contains the current value in the CDQP.



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:

```
oWorkQ.WorkItems.FilterExpression = "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 readme to determine if CR 12425 is implemented in your server.)

See “[Case Data vs. Work Item Data](#)” on page 91 for more information about the difference between Work Item Data and Case Data.

Using Work Queue Parameter Fields

Note - Previous versions of TIBCO iProcess Objects 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 of Miller, the **FilterExpression** property is set as follows:

```
oWorkQ.WorkItems.FilterExpression = "SW_QPARAM1?""Miller"""
```

This would be much more efficient than filtering on the LAST_NAME field.

The **SWWorkItem** object has four read-only properties that provide access to the values in the Work Queue Parameter fields — they are **WQParam1 - WQParam4**. These properties will contain the values you place in fields, SW_QPARAM1 - SW_QPARAM4, for each work item.

The **SWWorkQ** object has four read-only properties that contain a name for each of the Work Queue Parameter fields (WQParam1Name - WQParam4Name). If you use the TIBCO iProcess Client, 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 Client (Windows) Managers 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	<ul style="list-style-type: none"> • 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	<p>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.</p>

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

The FilterExpression property requires 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.

In **Visual Basic** you use the double quotes twice. In the example below, the two pairs of double quotes around LOAN signify that they are in reference to the string "LOAN", and not the ending quotes for the filter string.

```
oWorkQ.WorkItems.FilterExpression = "SW_PRONAME=""LOAN"""
```

In **Java** and **C++** you 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 setFilterExpression string.

```
oWorkQ.getWorkItems.setFilterExpression("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:

```
oWorkQ.WorkItems.FilterExpression = "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 97](#)).*

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 **SWNode** and **SWProc** objects contain methods that allow you to close or purge cases based on filter criteria. These methods are:

- **CloseByCriteria** - This method closes cases that match the specified filter criteria. To close a case, you must have system administrator authority (MENUNAME = ADMIN). See [“User Attributes” on page 221](#) for information about the MENUNAME attribute. You also cannot close a case from a slave node.
- **PurgeByCriteria** - This method purges cases that match the specified filter criteria. To purge a case, you must have system administrator authority (MENUNAME = ADMIN). See [“User Attributes” on page 221](#) for information about the MENUNAME attribute. 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.

How to Persist (Default) Filter Criteria

You can set *default* filter criteria for a work queue that persists on the queue. This causes future SWViews or SWXLists of work items on the current instance of the queue to use this default criteria.

Note - If you use the TIBCO iProcess Client, filter criteria that are defined on the Work Queue Manager Work Item List Filter dialog become the default filter criteria for that work queue. When an SWView or SWXList object is created for that work queue, the filter criteria defined on that dialog are written to the FilterExpression property.

The following methods on the **SWWorkQ** object 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):

- **SetDefCriteria** - This method sets the default filter and sort criteria for this work queue. It uses the current setting of the **FilterExpression** property and the **SortFields** property on the **SWView** in the **WorkItems** property in the current instance of the work queue to establish the default criteria.

Note that since this method uses the filter and sort criteria in the view in the WorkItems property, this method is practical to use if you are using SWViews, but not if you are using SWXLists. If you are using SWXLists, the SetDefCriteriaEx method is a better choice (see below).

- **SetDefCriteriaEx** - This method allows you to specify the default filter and sort criteria for this work queue by passing in the criteria as parameters. This causes the criteria you pass in this method to persist on this instance of the work queue, causing future SWViews or SWXLists of work items on this instance of the queue to use this default criteria.
- **ClearDefCriteria** - This method clears the default filter criteria that were set either through the Work Queue Manager or by using the **SetDefCriteria** or **SetDefCriteriaEx** methods (see above). (This also clears any default sort criteria that have been defined.)

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 call SetDefCriteria/SetDefCriteriaEx. The following are the filter criteria that are supported by the Work Queue Manager, that can, therefore, be persisted with the SetDefCriteria/SetDefCriteriaEx methods:

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

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

Also note the only equality operator that can be used in your filter expression when you are setting the default criteria with the SetDefCriteria/SetDefCriteriaEx methods is the '=' operator. The '<', '>', and '?' operators are not allowed (and since '?' is not allowed, no regular expression syntax can be used).

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 8
Only the WIS Work Item Filtering enhancement (CR 12744)	Chapter 9
Both the WIS Work Item Filtering and the Database Case Filtering enhancements (CRs 12744 and 13182)	Chapter 10

Introduction

You can *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 similar to the following:

```
oWorkQ.WorkItems.FilterExpression = "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 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.

To filter work items or cases, you must set the **FilterExpression** property equal to a filter expression string (as shown in the example above). The filter expression string is evaluated against each work item in the work queue or each case in the procedure, returning either True or False. If it returns True, the work item/case is included in the view/XList; if it returns False, the work item/case is not included in the view/XList.

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.).

How Filtering Differs Between Views and XLists

As described in the *Working With Lists* chapter, you may be using either **SWView** objects or **SWXList** objects to hold your work item and case objects. (Remember, all new development should make use of SWXLists because of their improved efficiency.) The way in which filter expressions are defined is somewhat different between these two objects. The differences are described below.

Defining Filter Expressions on SWView

The SWView object contains a **FilterExpression** property that is used to specify a Boolean expression that defines which cases (**SWCase** objects) or work items (**SWWorkItem** objects) for the respective procedure or work queue will be returned from the server and included in the view.

Each SWCase or SWWorkItem object is evaluated against the string expression specified in the **FilterExpression** property. Those that evaluate True are included in the view; those that evaluate False are not included in the view.

After setting or modifying the FilterExpression property, the **Rebuild** method must be called to update the view list based on the most recent filter criteria.

Number of Work Items or Cases in the Filtered View

The SWView object has a number of properties available that provide information about the number of objects in the view:

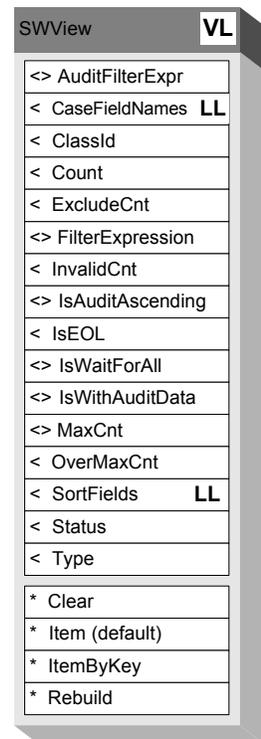
- **Count** - This property tells you the number of work items or cases that satisfied the filter expression and are currently in the view *at the client*. You can use this property to determine the total number of work items or cases available from the server. To do this you must iterate through all of the items in the view until **IsEOL** is True. See [“Determining the Total Number of Items Available” on page 71](#) for more information and an example.

Also see [“How SWViews are Created and Populated at the Client” on page 66](#) for information about how views are populated at the client.

- **ExcludeCnt** - This property contains the number of cases or work items that did not satisfy the Boolean expression specified in the FilterExpression property, and therefore, were not included in the view.
- **InvalidCnt** - When filtering cases, this property contains the number of cases not included in the view 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).

When filtering work items, this property is no longer applicable (it returns -1 if the view contains work items).

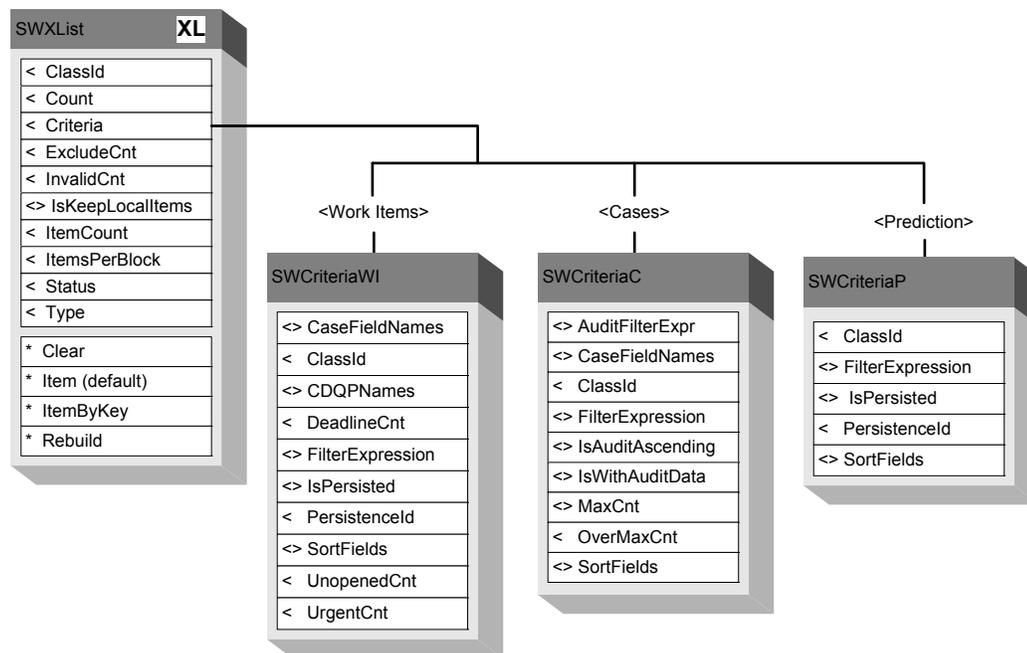
*The SWView object also contains an **AuditFilterExpr** property that is specific to filtering **SWAuditStep** objects that are in the **AuditSteps** list of the cases that are on the view. This filtering mechanism uses its own syntax and filtering criteria; it does not use the filter criteria defined in this chapter. See [“Filtering Audit Data” on page 245](#) for information about syntax and filter criteria specific to filtering audit steps.*



Defining Filter Expressions on SWXList

The SWXList object contains a **Criteria** property that points to an SWCriteriaWI, SWCriteriaC, or SWCriteriaP object, depending whether the XList contains cases, work items, or predicted work items:

- **SWCriteriaWI** - Contains properties that specify criteria for an XList that contains **SWWorkItem** objects.
- **SWCriteriaC** - Contains properties that specify criteria for an XList that contains **SWCase** objects.
- **SWCriteriaP** - Contains properties that specify criteria for an XList that contains **SWPredictedItem** objects. (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 152](#) for more information.)



The following properties are available on the objects shown above for use when filtering cases and work items on the XList:

- **Criteria** - This property contains a reference to the appropriate criteria object (SWCriteriaWI, SWCriteriaC, or SWCriteriaP), depending on whether the XList contains work items, cases, or predicted work items.
- **FilterExpression** - This read/write property is used to specify a Boolean expression that defines which work items (**SWWorkItem** objects), cases (**SWCase** objects), or predicted work items (**SWPredictedItem** objects) for the respective work queue or procedure will be returned from the server and included in the XList.

Each SWWorkItem, SWCase, or SWPredictedItem object is evaluated against the string expression specified in the FilterExpression property. Those that evaluate True are included in the XList; those that evaluate false are not included in the XList.

After setting or modifying the FilterExpression property, the **Rebuild** method must be called to update the XList list based on the most recent filter criteria.

Number of Work Items or Cases in the Filtered XList

The SWXList object has a number of properties available that provide information about the number of objects in the XList:

- **ItemCount** - This property contains the total number of work items or cases that satisfied the filter expression and are in the XList *at the server*. This count is available immediately after the XList is created (unlike on a view where you must iterate through the objects to determine the total number available).
- **Count** - This property tells you the number of work items or cases that satisfied the filter expression and are currently available in the XList *at the client*.

See “[How XLists are Created](#)” on page 75 for information about how XLists are populated with objects at the client.

- **ExcludeCnt** - This property contains the number of work items or cases that did not satisfy the Boolean expression specified in the FilterExpression property, and therefore, were not included in the XList.
- **InvalidCnt** - When filtering cases, this property contains the number of cases not included in the XList 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).

When filtering work items, this property is no longer applicable (it returns -1 if the XList contains work items).

*Note - The SWCriteriaC object also contains an **AuditFilterExpr** property that is specific to filtering **SWAuditStep** objects that are in the **AuditSteps** list of the cases that are on the XList. This filtering mechanism uses its own syntax and filtering criteria; it does not use the filter criteria defined in this chapter. See “[Filtering Audit Data](#)” on page 245 for information about syntax and filter criteria specific to filtering audit steps.*

Filtering/Sorting in an Efficient Manner

The way in which you write your filter expressions can have an effect 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.

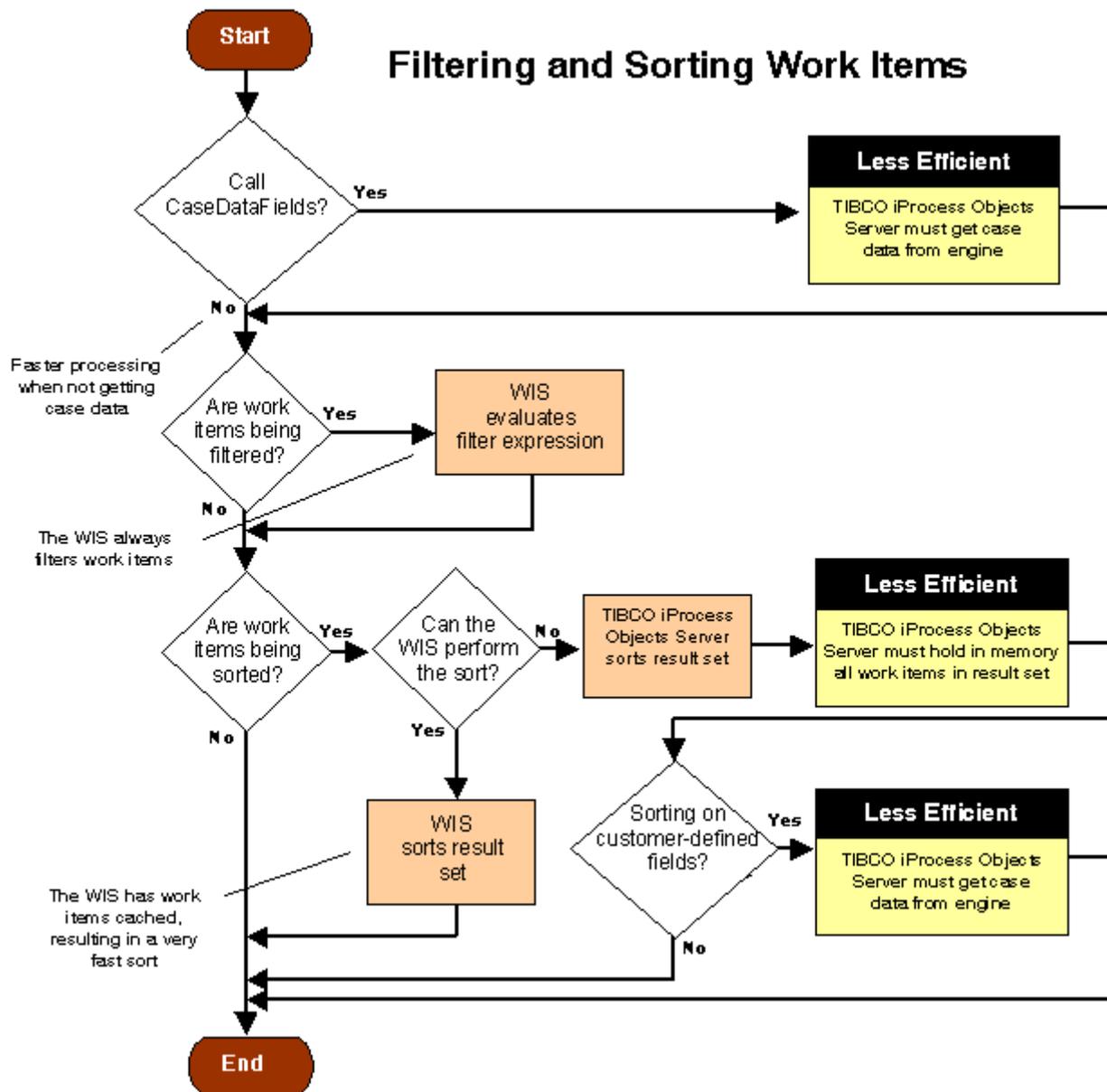
Flow diagrams (one for work items; one for cases) are shown in the following subsections that illustrate the decision process that takes place during a filter/sort operation. Note that the flow diagrams show filtering and sorting taking place in a single operation; that is the way filtering and sorting 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 133](#).
- 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:

- Calling **CaseFieldNames** - Adding field names to the **CaseFieldNames** property explicitly causes case data fields to be returned in the **Fields** property, which requires that the data be retrieved from the engine. See “[What is a Staffware Field?](#)” on page 86 for information about the use of **CaseFieldNames**.
- 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 call **CaseFieldNames** 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 147.)
Logical Operators	AND, OR
System Fields	All system fields that are applicable to work items (see the <i>Applies To</i> column in the table of system fields used for filtering — page 140)
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 144 for information.
Wild Cards	The wild card 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 149 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 147.

The following is an example of a filter expression for filtering work items:

- To define a filter for all unopened work items:

```
oCriteriaWI.FilterExpression = "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
<ul style="list-style-type: none"> • System fields that are “WIS-compatible”. See the WIS-compatible column in the table of <i>System Fields used in Sorting</i> on page 182. (The system fields must be applicable to filtering work items.) • Case Data Queue Parameter (CDQP) fields. See “Sorting on Case Data Fields” on page 184 for more information.
Sort Criteria the TIBCO iProcess Objects Server must Process
<ul style="list-style-type: none"> • System fields that are NOT “WIS-compatible”. See the WIS-compatible column in the table of <i>System Fields used in Sorting</i> on page 182. (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 184 for more information.

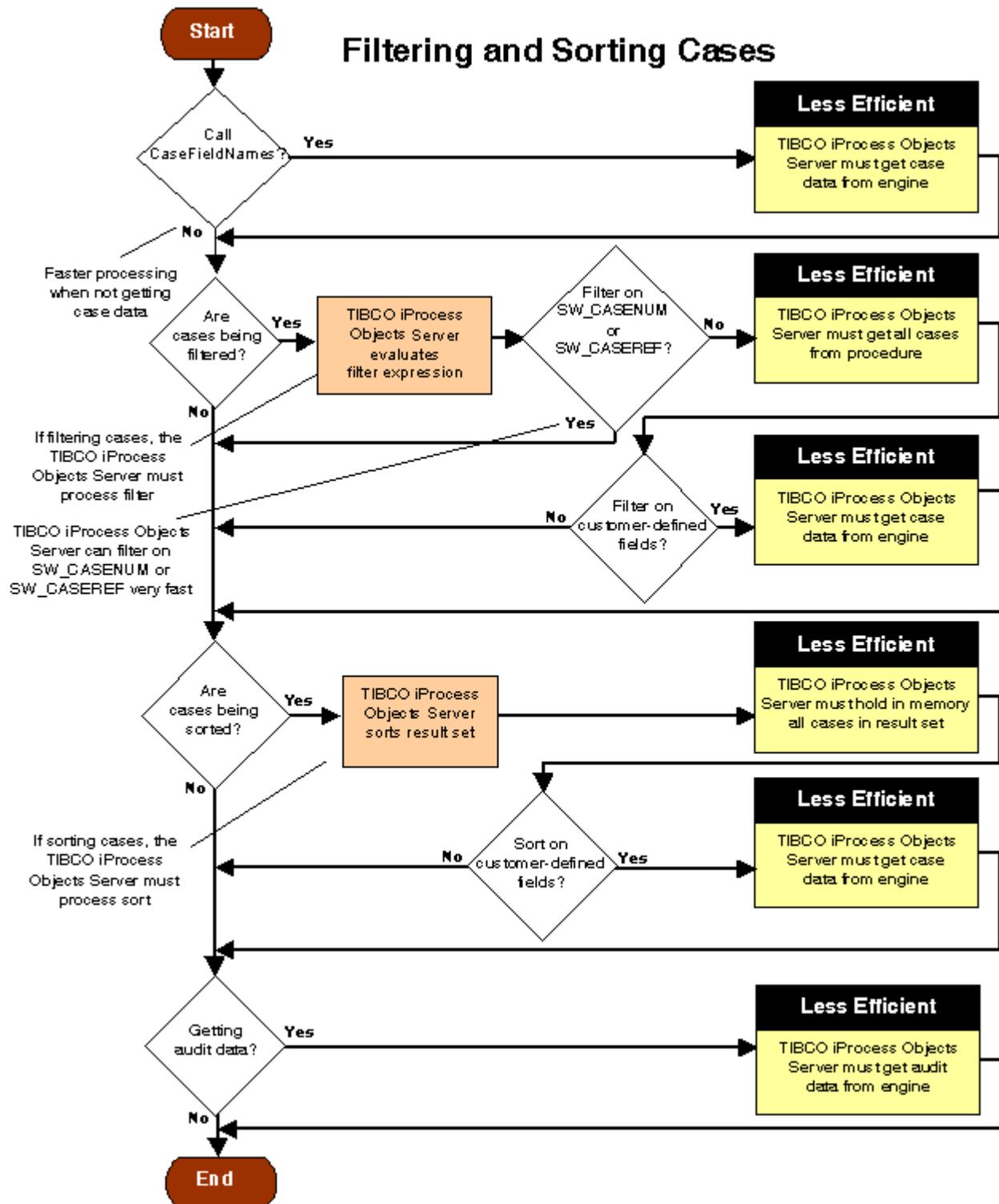
See “[Sorting Work Items and Cases](#)” on [page 178](#) for information about setting up sort criteria.

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 137](#) 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:

- Calling **CaseFieldNames** - Adding field names to the **CaseFieldNames** property explicitly causes case data fields to be returned in the **Fields** property, which requires that the data be retrieved from the engine. See [“What is a Staffware Field?” on page 86](#) for information about the use of **CaseFieldNames**.
- 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 and sorted *only* by the TIBCO iProcess Objects Server. This limits your options to perform an efficient filter and sort 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 147.)

Element	Description
System Fields	All system fields that are applicable to cases (see the <i>Applies To</i> column in the table of system fields used for filtering — page 140)
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 — see the illustration below.
Wild Cards	Note that the '*' and '?' characters are NOT interpreted as wild card 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 147 .

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):

```
oCriteriaC.FilterExpression = "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 examples of filtering on the case number and case reference number:

```
oProc.Cases.FilterExpression = "SW_CASENUM = 150"
```

```
oProc.Cases.FilterExpression = "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
<ul style="list-style-type: none"> • All system fields that are applicable to cases (see the <i>Applies To</i> column in the table of system fields used for sorting — page 182). • 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 the chapter, “[Sorting Work Items and Cases](#)” on [page 178](#) for specific information about setting up sort criteria.

Getting Audit Data

Getting audit data by setting the **IsWithAuditData** flag to True on the view or XList that holds your cases causes the TIBCO iProcess Objects Server to retrieve the audit data from the engine. This impacts the performance of a case filter operation.

Only include audit data in the cases in which it is needed. If you need it in all or most of the cases in the view/XList, set **IsWithAuditData** on the view/XLists. If it is needed on only one or a few cases, request it on those specific cases by setting the **IsWithAuditData** flag only on those cases.

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 140 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 client through properties on `SWWorkItem` and `SWCase`. For example, `SW_CASENUM` is available to the client in the `SWCase.CaseNumber` property. The engine, however, doesn't have access to those properties, so the property names from `SWWorkItem` and `SWCase` can't be used in filter and sort criteria — instead, the system field names need to be used in your expressions. For example:

```
oWorkQ.WorkItems.FilterExpression = "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_DEADLINE_DATE	Date	10	X	
Deadline expired flag (1 - expired; 0 - not expired)	SW_EXPIRED	Numeric	1	X	
Deadline set flag (1 - has deadline; 0 - does not have deadline)	SW_HASDEADLINE	Numeric	1	X	
Deadline time	SW_DEADLINE_TIME	Time	5	X	

Filter Criteria	System Field	Data Type	Length	Applies to work items	Applies to cases
Forwardable work item flag (1 - forwardable; 0 - not forwardable)	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	
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	7	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 open)	SW_NEW	Numeric	1	X	
Urgent flag (1- urgent; 0 - not 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 97). 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 97). 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 Staffware 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.

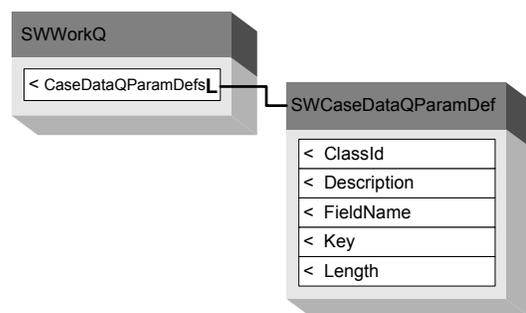
Note - With the WIS work item filtering enhancement, case data fields can be included in filter expressions only if they are defined as Case Data Queue Parameter (CDQP) fields. If your filter expression references a field that is not a CDQP for the queue, the WIS will return a syntax error, which causes 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

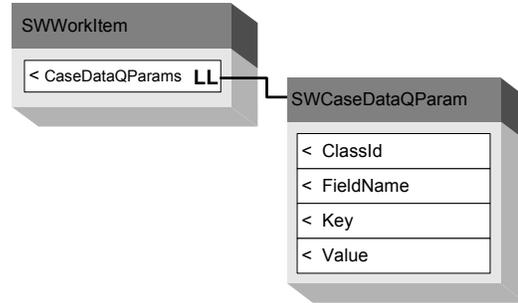
Case Data Queue Parameter (CDQP) fields provide an efficient method of filtering on the value of fields in your work items. To make use of this functionality, you must first pre-designate the fields you want to filter on as CDQP fields. Fields are designated as CDQP fields with the utility, **swutil**. 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. See the *TIBCO iProcess Engine Administrator's Guide* for information about using **swutil**.

*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 in the **SWWorkQ.CaseDataQParamDefs** property. This property contains a list of **SWCaseDataQParamDef** objects, one for each case data field that has been designated as a CDQP field in the work queue. This list tells you the CDQPs that are available for filter and sort criteria.



The **CaseDataQParams** property on **SWorkItem** provides access to CDQPs that are being used in the work item. Note, however, to control resource usage, you can specify which CDQPs to return from the server by using the **CDQPNames** property on the **SWCriteriaWI** object. By default, all CDQPs used in a work item are returned from the server. The **SWCaseDataQParam** object contains the current value in the CDQP.



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:

```
oWorkQ.WorkItems.FilterExpression = "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 91](#) for more information.

Using Work Queue Parameter Fields

Note - Previous versions of TIBCO iProcess Objects 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:

```
oWorkQ.WorkItems.FilterExpression = "SW_QPARAM1?"Miller""
```

This would be much more efficient than filtering on the LAST_NAME field.

The **SWWorkItem** object has four read-only properties that provide access to the values in the Work Queue Parameter fields — they are **WQParam1 - WQParam4**. These properties will contain the values you place in fields, SW_QPARAM1 - SW_QPARAM4, for each work item.

The **SWWorkQ** object has four read-only properties that contain a name for each of the Work Queue Parameter fields (WQParam1Name - WQParam4Name). If you use the TIBCO iProcess Client, 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 Client (Windows) Managers 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	<ul style="list-style-type: none"> • 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	<p>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.</p>

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 wild-card 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 an TIBCO iProcess Objects Server that does not have the WIS work item filtering enhancement (CR 12744) to one that does (see [page 126](#) 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.

Using Escape Characters in the Filter Expression

The FilterExpression property requires 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.

In **Visual Basic** you use the double quotes twice. In the example below, the two pairs of double quotes around LOAN signify that they are in reference to the string "LOAN", and not the ending quotes for the filter string.

```
oWorkQ.WorkItems.FilterExpression = "SW_PRONAME=""LOAN"""
```

In **Java** and **C++** you 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 setFilterExpression string.

```
oWorkQ.getWorkItems.setFilterExpression("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:

```
oWorkQ.WorkItems.FilterExpression = "SOC_SEC_NUM=SW_NA"
```

- compare the field to an empty set of quotes. For example:

```
oWorkQ.WorkItems.FilterExpression = "SOC_SEC_NUM=""
```

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 97](#)).*

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 **SWNode** and **SWProc** objects contain methods that allow you to close or purge cases based on filter criteria. These methods are:

- **CloseByCriteria** - This method closes cases that match the specified filter criteria. To close a case, you must have system administrator authority (MENUNAME = ADMIN). See “[User Attributes](#)” on page 221 for information about the MENUNAME attribute. You also cannot close a case from a slave node.
- **PurgeByCriteria** - This method purges cases that match the specified filter criteria. To purge a case, you must have system administrator authority (MENUNAME = ADMIN). See “[User Attributes](#)” on page 221 for information about the MENUNAME attribute. 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.

How to Persist (Default) Filter Criteria

You can set *default* filter criteria for a work queue that persists on the queue. This causes future SWViews or SWXLists of work items on the current instance of the queue to use this default criteria.

Note - If you use the TIBCO iProcess Client, filter criteria that are defined on the Work Queue Manager Work Item List Filter dialog become the default filter criteria for that work queue. When an SWView or SWXList object is created for that work queue, the filter criteria defined on that dialog are written to the FilterExpression property.

The following methods on the **SWWorkQ** object 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):

- **SetDefCriteria** - This method sets the default filter and sort criteria for this work queue. It uses the current setting of the **FilterExpression** property and the **SortFields** property on the **SWView** in the **WorkItems** property in the current instance of the work queue to establish the default criteria.

Note that since this method uses the filter and sort criteria in the view in the WorkItems property, this method is practical to use if you are using SWViews, but not if you are using SWXLists. If you are using SWXLists, the SetDefCriteriaEx method is a better choice (see below).

- **SetDefCriteriaEx** - This method allows you to specify the default filter and sort criteria for this work queue by passing in the criteria as parameters. This causes the criteria you pass in this method to persist on this instance of the work queue, causing future SWViews or SWXLists of work items on this instance of the queue to use this default criteria.

- **ClearDefCriteria** - This method clears the default filter criteria that were set either through the Work Queue Manager or by using the **SetDefCriteria** or **SetDefCriteriaEx** methods (see above). (This also clears any default sort criteria that have been defined.)

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 call **SetDefCriteria/SetDefCriteriaEx**. The following are the filter criteria that are supported by the Work Queue Manager, that can, therefore, be persisted with the **SetDefCriteria/SetDefCriteriaEx** methods:

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 only equality operator that can be used in your filter expression when you are setting the default criteria with the **SetDefCriteria/SetDefCriteriaEx** methods is the '=' operator. The '<', '>', and '?' operators are not allowed (and since '?' is not allowed, no regular expression syntax can be used).

10

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 8
Only the WIS Work Item Filtering enhancement (CR 12744)	Chapter 9
Both the WIS Work Item Filtering and the Database Case Filtering enhancements (CRs 12744 and 13182)	Chapter 10

Introduction

You can *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 similar to the following:

```
oWorkQ.WorkItems.FilterExpression = "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 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.

To filter work items or cases, you must set the **FilterExpression** property equal to a filter expression string (as shown in the example above). The filter expression string is evaluated against each work item in the work queue or each case in the procedure, returning either True or False. If it returns True, the work item/case is included in the view/XList; if it returns False, the work item/case is not included in the view/XList.

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.).

How Filtering Differs Between Views and XLists

As described in the *Working With Lists* chapter, you may be using either **SWView** objects or **SWXList** objects to hold your work item and case objects. (Remember, all new development should make use of SWXLists because of their improved efficiency.) The way in which filter expressions are defined is somewhat different between these two objects. The differences are described below.

Defining Filter Expressions on SWView

The SWView object contains a **FilterExpression** property that is used to specify a Boolean expression that defines which cases (**SWCase** objects) or work items (**SWWorkItem** objects) for the respective procedure or work queue will be returned from the server and included in the view.

Each SWCase or SWWorkItem object is evaluated against the string expression specified in the **FilterExpression** property. Those that evaluate True are included in the view; those that evaluate False are not included in the view.

After setting or modifying the FilterExpression property, the **Rebuild** method must be called to update the view list based on the most recent filter criteria.

Number of Work Items or Cases in the Filtered View

The SWView object has a number of properties available that provide information about the number of objects in the view:

- **Count** - This property tells you the number of work items or cases that satisfied the filter expression and are currently in the view *at the client*. You can use this property to determine the total number of work items or cases available from the server. To do this you must iterate through all of the items in the view until **IsEOL** is True. See [“Determining the Total Number of Items Available” on page 71](#) for more information and an example.

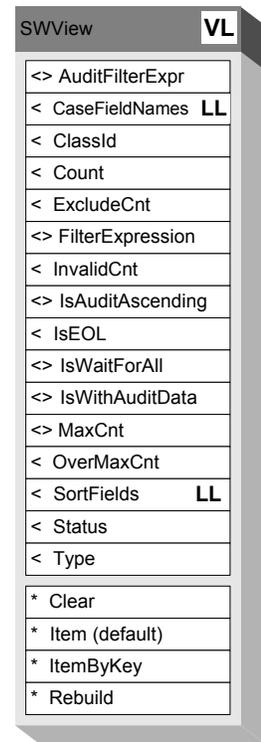
Also see [“How SWViews are Created and Populated at the Client” on page 66](#) for information about how views are populated at the client.

- **ExcludeCnt** - When filtering work items, this property contains the number of work items that did not satisfy the Boolean expression specified in the FilterExpression property, and therefore, were not included in the view.

When filtering cases, this property is no longer applicable (it returns -1 if the view 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.

- **InvalidCnt** - This count is no longer applicable when filtering work items or cases (it always returns -1).

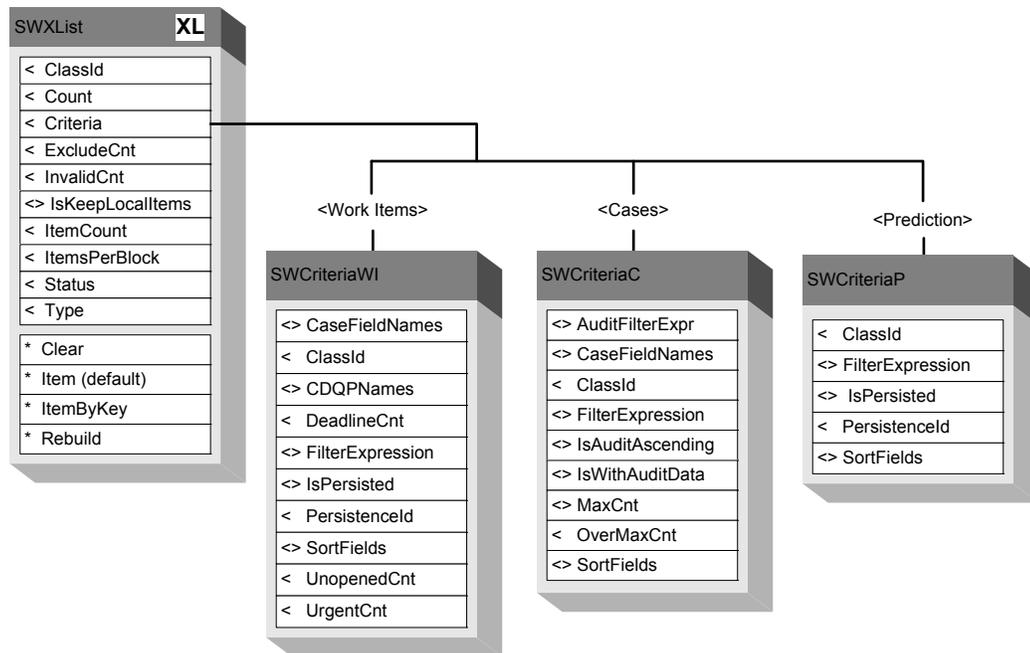
The SWView object also contains an **AuditFilterExpr** property that is specific to filtering **SWAuditStep** objects that are in the **AuditSteps** list of the cases that are on the view. This filtering mechanism uses its own syntax and filtering criteria; it does not use the filter criteria defined in this chapter. See [“Filtering Audit Data” on page 245](#) for information about syntax and filter criteria specific to filtering audit steps.



Defining Filter Expressions on SWXList

The SWXList object contains a **Criteria** property that points to an SWCriteriaWI, SWCriteriaC, or SWCriteriaP object, depending whether the XList contains cases, work items, or predicted work items:

- **SWCriteriaWI** - Contains properties that specify criteria for an XList that contains **SWWorkItem** objects.
- **SWCriteriaC** - Contains properties that specify criteria for an XList that contains **SWCase** objects.
- **SWCriteriaP** - Contains properties that specify criteria for an XList that contains **SWPredictedItem** objects. (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 160 for more information.)



The following properties are available on the objects shown above for use when filtering cases and work items on the XList:

- **Criteria** - This property contains a reference to the appropriate criteria object (SWCriteriaWI, SWCriteriaC, or SWCriteriaP), depending on whether the XList contains work items, cases, or predicted work items.
- **FilterExpression** - This read/write property is used to specify a Boolean expression that defines which work items (**SWWorkItem** objects), cases (**SWCase** objects), or predicted work items (**SWPredictedItem** objects) for the respective work queue or procedure will be returned from the server and included in the XList.

Each SWWorkItem, SWCase, or SWPredictedItem object is evaluated against the string expression specified in the FilterExpression property. Those that evaluate True are included in the XList; those that evaluate false are not included in the XList.

After setting or modifying the FilterExpression property, the **Rebuild** method must be called to update the XList list based on the most recent filter criteria.

Number of Work Items or Cases in the Filtered XList

The SWXList object has a number of properties available that provide information about the number of objects in the XList:

- **ItemCount** - This property contains the total number of work items or cases that satisfied the filter expression and are in the XList *at the server*. This count is available immediately after the XList is created (unlike on a view where you must iterate through the objects to determine the total number available).
- **Count** - This property tells you the number of work items or cases that satisfied the filter expression and are currently available in the XList *at the client*.

See “[How XLists are Created](#)” on page 75 for information about how XLists are populated with objects at the client.

- **ExcludeCnt** - When filtering work items, this property contains the number of work items that did not satisfy the Boolean expression specified in the FilterExpression property, and therefore, were not included in the XList.

When filtering cases, this property is no longer applicable (it returns -1 if the XList 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.

- **InvalidCnt** - This count is no longer applicable when filtering work items or cases (it always returns -1).

*Note - The SWCriteriaC object also contains an **AuditFilterExpr** property that is specific to filtering SWAuditStep objects that are in the **AuditSteps** list of the cases that are on the XList. This filtering mechanism uses its own syntax and filtering criteria; it does not use the filter criteria defined in this chapter. See “[Filtering Audit Data](#)” on page 245 for information about syntax and filter criteria specific to filtering audit steps.*

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 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 in UNIX

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*.

Filtering/Sorting in an Efficient Manner

The way in which you write your filter expressions can have an effect 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.

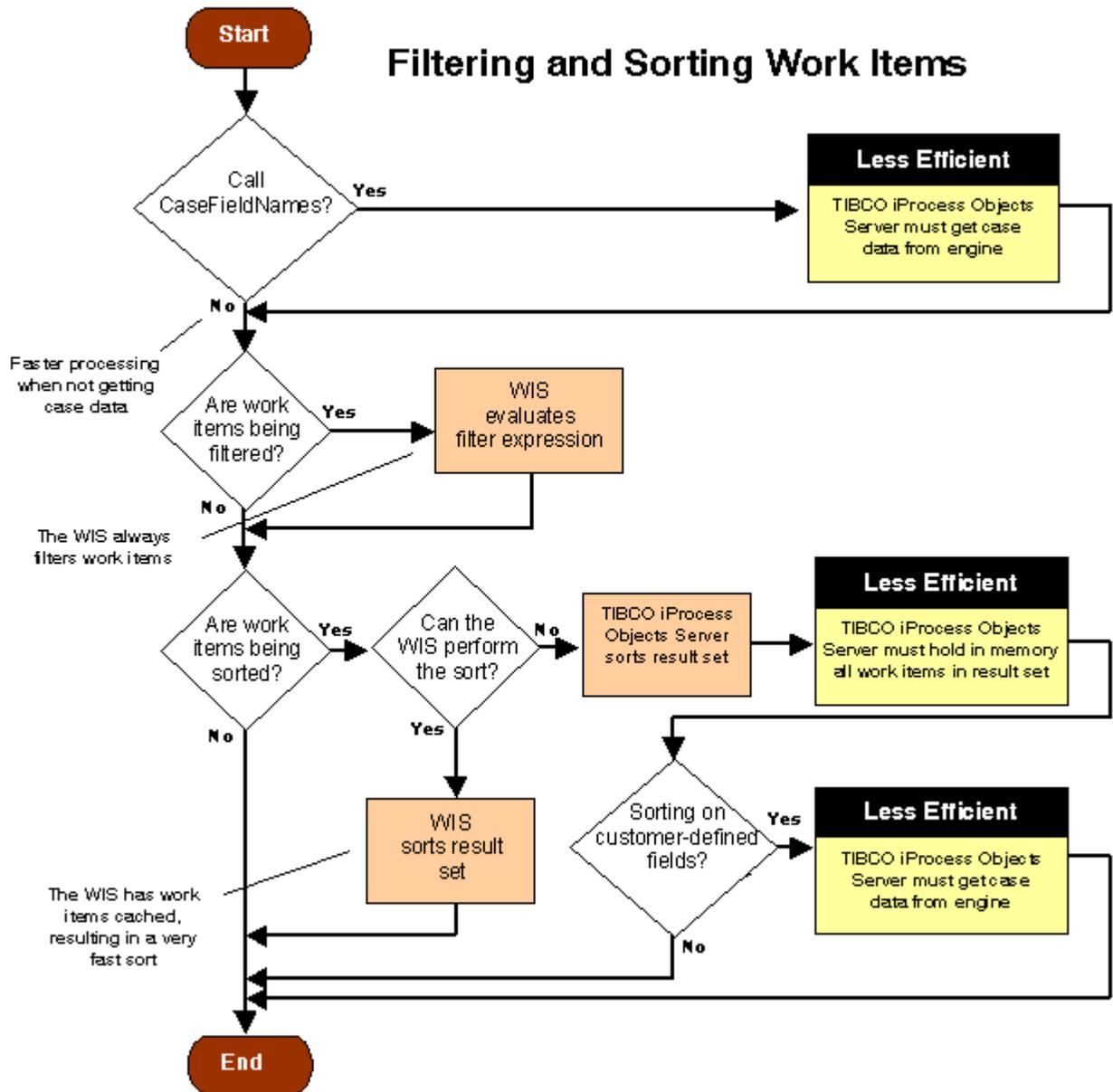
Flow diagrams (one for work items; one for cases) are shown in the following subsections that illustrate the decision process that takes place during a filter/sort operation. Note that the flow diagrams show filtering and sorting taking place in a single operation; that is the way filtering and sorting 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 the table 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.
- 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:

- Calling **CaseFieldNames** - Adding field names to the **CaseFieldNames** property explicitly causes case data fields to be returned in the **Fields** property, which requires that the data be retrieved from the engine. See “[What is a Staffware Field?](#)” on page 86 for information about the use of **CaseFieldNames**.
- 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 TIBCO iProcess 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 call **CaseFieldNames** 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 172.)
Logical Operators	AND, OR
System Fields	All system fields that are applicable to work items (see the <i>Applies To</i> column in the table of system fields used for filtering — page 165).
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 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 169 for information.
Wild Cards	The wild card 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 175 for information.

Element	Description
Regular Expression	Regular expressions can be used when filtering work items, allowing you to do complex pattern matching. See “Using Regular Expressions” on page 172 .

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:

```
oCriteriaWI.FilterExpression = "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:

```
oCriteriaWI.FilterExpression = "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
<ul style="list-style-type: none"> System fields that are “WIS-compatible”. See the WIS-compatible column in the table of <i>System Fields used in Sorting</i> on page 182. (The system fields must be applicable to filtering work items.) Case Data Queue Parameter (CDQP) fields. See “Sorting on Case Data Fields” on page 184 for more information.
Sort Criteria the TIBCO iProcess Objects Server must Process
<ul style="list-style-type: none"> System fields that are NOT “WIS-compatible”. See the WIS-compatible column in the table of <i>System Fields used in Sorting</i> on page 182. (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 184 for more information.

See the chapter, [“Sorting Work Items and Cases” on page 178](#) for information about setting up sort criteria.

Filtering/Sorting Cases

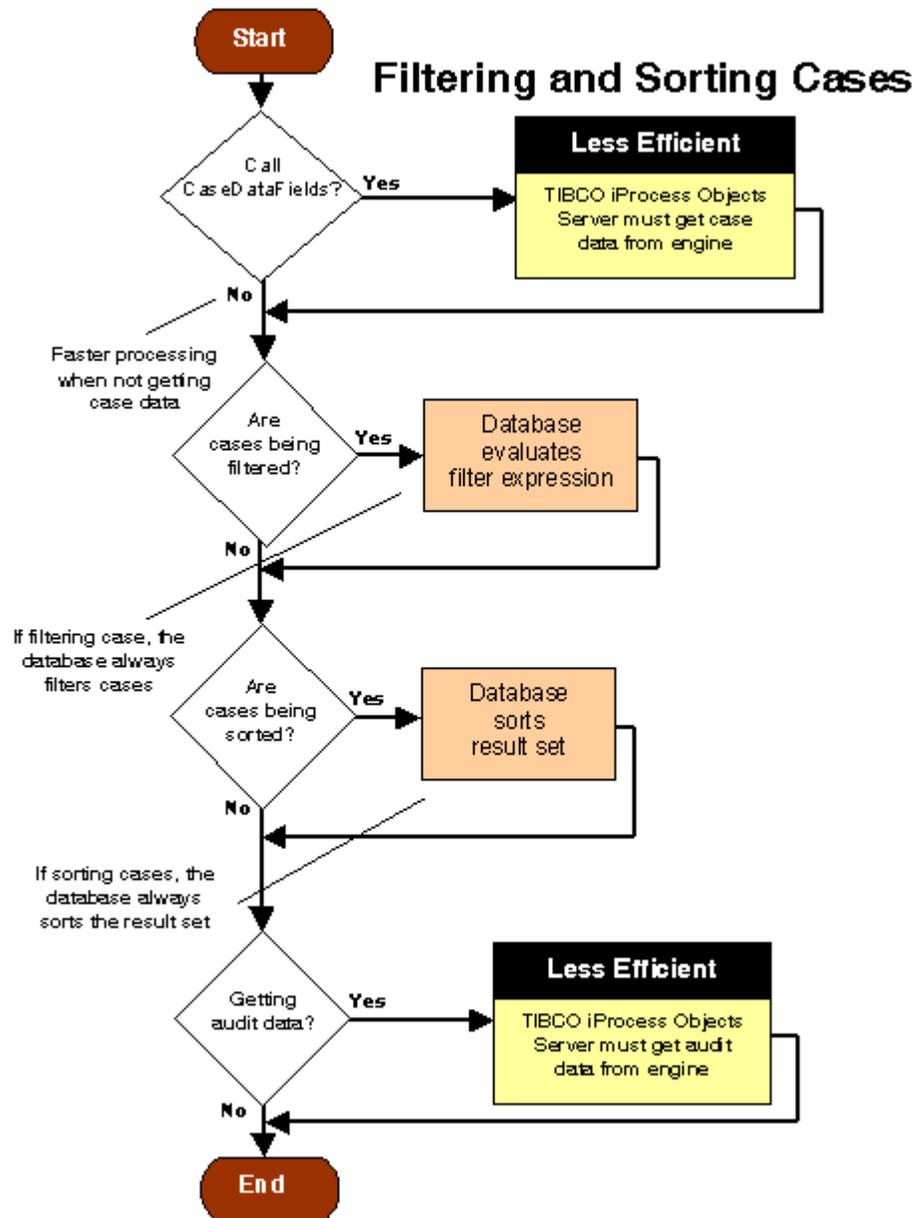
When filtering and sorting cases:

- 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 162.

- 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 163 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 cause the TIBCO iProcess Objects Server to get case data:

- Calling **CaseFieldNames** - Adding field names to the **CaseFieldNames** property explicitly causes case data fields to be returned in the **Fields** property, which requires that the data be retrieved from the engine. See “[What is a Staffware Field?](#)” on page 86 for information about the use of **CaseFieldNames**.

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 172.)
System Fields	All system fields that are applicable to cases (see the <i>Applies To</i> column in the table of system fields used for filtering — page 165).
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 are not supported, e.g., FIELD1 = FIELD2, FIELD2 > FIELD3, etc.).
Wild Cards	The wild card 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 172.

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:

```
oCriteriaC.FilterExpression = "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":

```
oCriteriaC.FilterExpression = "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 182](#)).
- Case data fields can be included in your sort criteria when sorting cases.

See the chapter, “[Sorting Work Items and Cases](#)” on [page 178](#) for specific information about setting up sort criteria.

Getting Audit Data

Getting audit data by setting the **IsWithAuditData** flag to True on the view or XList that holds your cases causes the TIBCO iProcess Objects Server to retrieve the audit data from the engine. This impacts the performance of a case filter operation.

Only include audit data in the cases in which it is needed. If you need it in all or most of the cases in the view/XList, set **IsWithAuditData** on the view/XLists. If it is needed on only one or a few cases, request it on those specific cases by setting the **IsWithAuditData** flag only on those cases.

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 165* 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 client through properties on SWWorkItem and SWCase. For example, **SW_CASENUM** is available to the client in the **SWCase.CaseNumber** property. The engine, however, doesn't have access to those properties, so the property names from SWWorkItem and SWCase can't be used in filter and sort criteria — instead, the system field names need to be used in your expressions. For example:

```
oWorkQ.WorkItems.FilterExpression = "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

Filter Criteria	System Field	Data Type	Length	Applies to work items	Applies to cases
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_DEADLINE_DATE	Date	10	X	
Deadline expired flag (1 - expired; 0 - not expired)	SW_EXPIRED	Numeric	1	X	
Deadline set flag (1 - has deadline; 0 - does not have deadline)	SW_HASDEADLINE	Numeric	1	X	
Deadline time	SW_DEADLINE_TIME	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	7	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	

Filter Criteria	System Field	Data Type	Length	Applies to work items	Applies to cases
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 - have been open)	SW_NEW	Numeric	1	X	
Urgent flag (1- urgent; 0 - not 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 97). 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 97). 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

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.

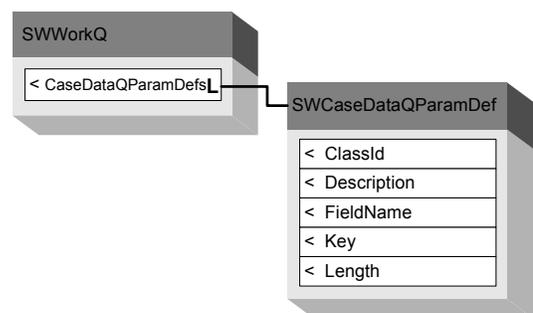
Note - With the WIS work item filtering enhancement, case data fields can be included in filter expressions only if they are defined as Case Data Queue Parameter (CDQP) fields. If your filter expression references a field that is not a CDQP for the queue, the WIS will return a syntax error, which causes 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

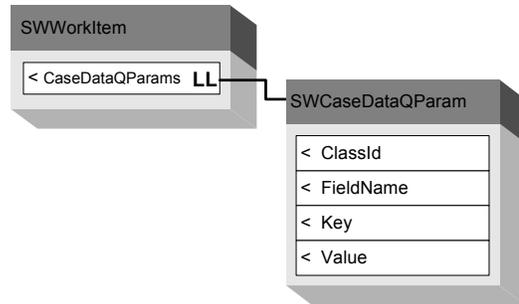
Case Data Queue Parameter (CDQP) fields provide an efficient method of filtering on the value of fields in your work items. To make use of this functionality, you must first pre-designate the fields you want to filter on as CDQP fields. Fields are designated as CDQP fields with the utility, **swutil**. 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. See the *TIBCO iProcess Engine Administrator's Guide* for information about using **swutil**.

*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 in the **SWWorkQ.CaseDataQParamDefs** property. This property contains a list of **SWCaseDataQParamDef** objects, one for each case data field that has been designated as a CDQP field in the work queue. This list tells you the CDQPs that are available for filter and sort criteria.



The **CaseDataQParams** property on **SWorkItem** provides access to CDQPs that are being used in the work item. Note, however, to control resource usage, you can specify which CDQPs to return from the server by using the **CDQPNames** property on the **SWCriteriaWI** object. By default, all CDQPs used in a work item are returned from the server. The **SWCaseDataQParam** object contains the current value in the CDQP.



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:

```
oWorkQ.WorkItems.FilterExpression = "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 91 for more information.

Using Work Queue Parameter Fields

Note - Previous versions of TIBCO iProcess Objects 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:

```
oWorkQ.WorkItems.FilterExpression = "SW_QPARAM1?"Miller""
```

This would be much more efficient than filtering on the LAST_NAME field.

The **SWWorkItem** object has four read-only properties that provide access to the values in the Work Queue Parameter fields — they are **WQParam1 - WQParam4**. These properties will contain the values you place in fields, SW_QPARAM1 - SW_QPARAM4, for each work item.

The **SWWorkQ** object has four read-only properties that contain a name for each of the Work Queue Parameter fields (WQParam1Name - WQParam4Name). If you use the TIBCO iProcess Client, 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 Client (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	<ul style="list-style-type: none"> • 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 wild-card 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 an TIBCO iProcess Objects Server that does not have the WIS work item filtering enhancement (CR 12744) to one that does (see [page 152](#) 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.

Note - If you are moving from an TIBCO iProcess Objects Server that does not have the database case filtering enhancement (CR 13182) to one that does (see [page 152](#) 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

The FilterExpression property requires 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.

In **Visual Basic** you use the double quotes twice. In the example below, the two pairs of double quotes around LOAN signify that they are in reference to the string "LOAN", and not the ending quotes for the filter string.

```
oWorkQ.WorkItems.FilterExpression = "SW_PRONAME=""LOAN"""
```

In **Java** and **C++** you 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 setFilterExpression string.

```
oWorkQ.getWorkItems.setFilterExpression("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:

```
oWorkQ.WorkItems.FilterExpression = "SOC_SEC_NUM=SW_NA"
```

- compare the field to an empty set of quotes. For example:

```
oWorkQ.WorkItems.FilterExpression = "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 97](#)).*

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 **SWNode** and **SWProc** objects contain methods that allow you to close or purge cases based on filter criteria. These methods are:

- **CloseByCriteria** - This method closes cases that match the specified filter criteria. To close a case, you must have system administrator authority (MENUNAME = ADMIN). See “[User Attributes](#)” on page 221 for information about the MENUNAME attribute. You also cannot close a case from a slave node.
- **PurgeByCriteria** - This method purges cases that match the specified filter criteria. To purge a case, you must have system administrator authority (MENUNAME = ADMIN). See “[User Attributes](#)” on page 221 for information about the MENUNAME attribute. 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.

How to Persist (Default) Filter Criteria

You can set *default* filter criteria for a work queue; this criteria persists on the queue. This causes future SWViews or SWXLists of work items on the current instance of the queue to use this default criteria.

Note - If you use the TIBCO iProcess Client, filter criteria that are defined on the Work Queue Manager Work Item List Filter dialog become the default filter criteria for that work queue. When an SWView or SWXList object is created for that work queue, the filter criteria defined on that dialog are written to the FilterExpression property.

The following methods on the **SWWorkQ** object 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):

- **SetDefCriteria** - This method sets the default filter and sort criteria for this work queue. It uses the current setting of the **FilterExpression** property and the **SortFields** property on the **SWView** in the **WorkItems** property in the current instance of the work queue to establish the default criteria.

Note that since this method uses the filter and sort criteria in the view in the WorkItems property, this method is practical to use if you are using SWViews, but not if you are using SWXLists. If you are using SWXLists, the SetDefCriteriaEx method is a better choice (see below).

- **SetDefCriteriaEx** - This method allows you to specify the default filter and sort criteria for this work queue by passing in the criteria as parameters. This causes the criteria you pass in this method to persist on this instance of the work queue, causing future SWViews or SWXLists of work items on this instance of the queue to use this default criteria.
- **ClearDefCriteria** - This method clears the default filter criteria that were set either through the Work Queue Manager or by using the **SetDefCriteria** or **SetDefCriteriaEx** methods (see above). (This also clears any default sort criteria that have been defined.)

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 call SetDefCriteria/SetDefCriteriaEx. The following are the filter criteria that are supported by the Work Queue Manager, that can, therefore, be persisted with the SetDefCriteria/SetDefCriteriaEx methods:

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 only equality operator that can be used in your filter expression when you are setting the default criteria with the SetDefCriteria/SetDefCriteriaEx methods is the '=' operator. The '<', '>', and '?' operators are not allowed (and since '?' is not allowed, no regular expression syntax can be used).

Sorting Work Items and Cases

Introduction

You can *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).

How Sorting Differs Between Views and XLists

As described in the *Working With Lists* chapter, you may be using either **SWView** objects or **SWXList** objects to hold your work item and case objects. (All new development should make use of SWXLists because of their improved efficiency.) The way in which sort criteria are defined is somewhat different between these two objects. The differences are described below.

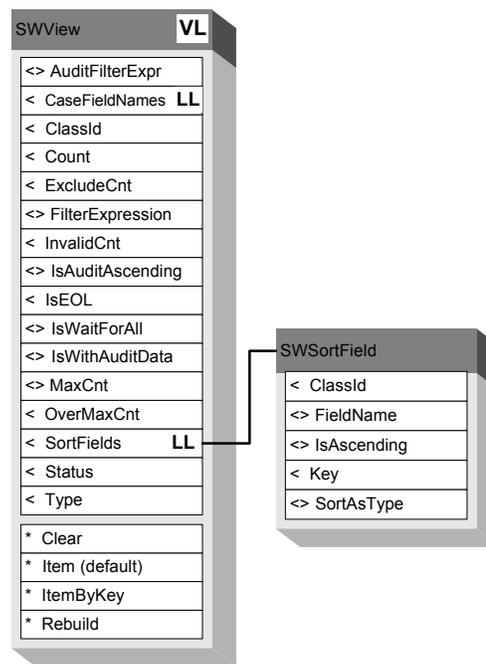
Defining Sort Criteria on SWView

Work items and cases in a view are sorted by using the **SortFields** property on the SWView object. SortFields is a local list of **SWSortField** objects, each representing one field (system field or case data field — described in the following subsections) on which the work items or cases in the view are sorted.

The order in which the SWSortField objects are placed in the SortFields local list defines the order in which the work items or cases are sorted in the view.

SWSortField objects are added to the SortFields local list by using the **Add** method on **SWLocList**.

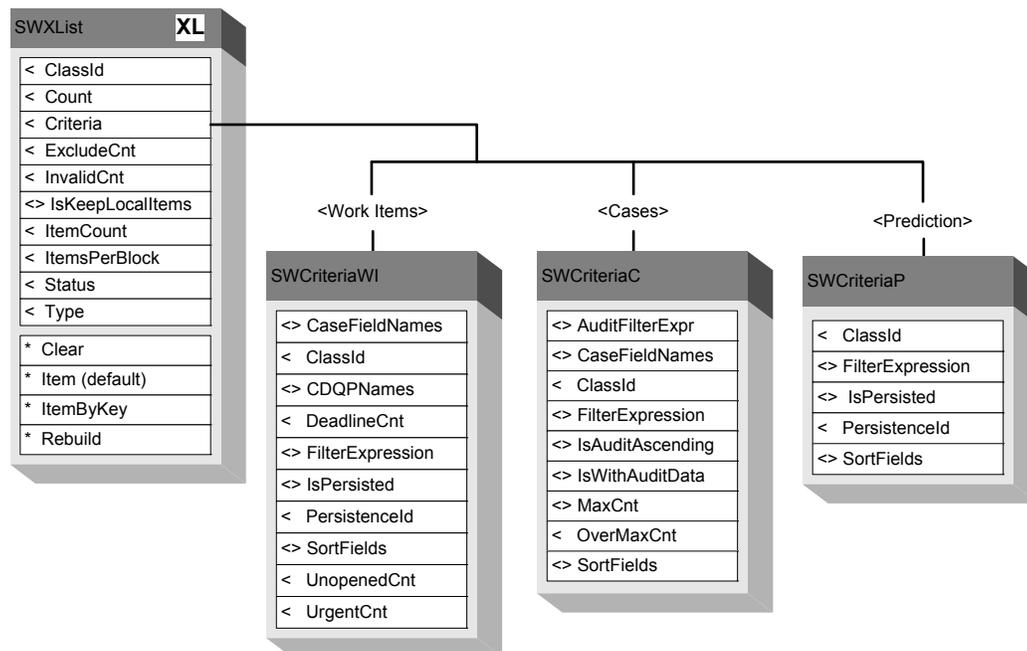
After setting or modifying the SortFields local list, the **Rebuild** method on SWView must be invoked to update the view based on the most recent sort criteria.



Defining Sort Criteria on SWXList

The SWXList object contains a **Criteria** property that points to an **SWCriteriaWI**, **SWCriteriaC**, or **SWCriteriaP** object, depending whether the XList contains work items, cases, or predicted work items, respectively:

- **SWCriteriaWI** - Contains properties that specify criteria for an XList that contains **SWWorkItem** objects.
- **SWCriteriaC** - Contains properties that specify criteria for an XList that contains **SWCase** objects.
- **SWCriteriaP** - Contains properties that specify criteria for an XList that contains **SWPredictedItem** objects.



Work items, cases, and predicted work items in an XList are sorted by using the **SortFields** property on the SWCriteriaWI object (if the XList contains work items), SWCriteriaC object (if the XList contains cases), or SWCriteriaP (if the XList contains predicted work items).

The SortFields property is a read/write array of **SWSortField** objects, each representing one field (system field or case data field — described in the following subsections) on which the work items, cases, or predicted work items in the XList are sorted.

The order in which the SWSortField objects are placed in the SortFields array defines the order in which the work items, cases, or predicted work items are sorted in the XList.

After setting or modifying the SortFields array, the **Rebuild** method on SWXList must be invoked to update the XList based on the most recent sort criteria.

Specifying Sort Criteria

You specify how you want the cases or work items in the view or XList to be sorted by adding one or more **SWSortField** objects to the **SortFields** local list (on a view) or array (on an XList). The **SWSortField** object represents the *field* on which the list of cases or work items are sorted. The fields on which it's sorted can be:

- **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 182 for more information.
- **Case Data Fields** - These are fields displayed on the Staffware form. You can sort according to the value of these fields. See “[Sorting on Case Data Fields](#)” on page 184 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.

Important - The way in which you specify sort criteria in your client application can greatly affect the speed at which the sorting is processed. You are strongly encouraged to read the *Filtering Work Items and Cases* chapter that is applicable to you (depending on the filtering enhancements you have in your TIBCO iProcess Objects Server). The *Filtering Work Items and Cases* chapters provide guidelines about how to specify filter and sort criteria to ensure an efficient filter/sort operation.

The following are a couple of examples of how the **SortFields** property is used to specify sort criteria.

Example of Sorting on a View:

```
Set oWorkItems = oWorkQ.WorkItems
oWorkItems.Clear
oWorkItems.MaxCnt = 20
Set oWorkItem = oWorkItems(0)      ' msg sent since first access after clear
Debug.Print "Field Count (for workitem:" & oWorkItem.key & ") = "_
    & oWorkItem.Case.Fields.Count    ' No fields returned for Case associated
    ' with Workitem

oWorkItems.CaseFieldNames.Clear     ' ensure empty local list before adding
oWorkItems.CaseFieldNames.Add "SW_CASENUM" ' return Case Number Field
oWorkItems.CaseFieldNames.Add "TESTPROFIELD3" ' return User defined Field

    ' Enable workitems to be sorted by Casenumber
For Each oSortField In oWorkItems.SortFields ' Note there are sortfields present
    ' by default
    Debug.Print "Sortfield Field Name = " & oSortField.FieldName
Next

'clear current list since want to sort ONLY by casenum
oWorkItems.SortFields.Clear

' Create and configure Sortfield
Set oSortField = New SWSortField
oSortField.FieldName = "SW_CASENUM"
oSortField.IsAscending = False
oSortField.SortAsType = swNumericSort
' Add SortField to local list
oWorkItems.SortFields.Add oSortField
```

See [page 322](#) for a comprehensive example.

Note that the maximum number of sort fields that can be set is 10. Any additional will be ignored.

Example of Sorting on an XList:

```

Set oWorkItemsX = oWorkQ.WorkItemsX
Set oWorkItem = oWorkItemsX(0)      ' msg sent since first access after clear
' No fields returned for Case associated with Workitem
Debug.Print ("Field Count (for workitem:" & oWorkItem.key & ") = "_
            & oWorkItem.Case.Fields.Count)
ReDim Fieldnames(1)
Fieldnames(0) = "SW_CASENUM"        ' return Case Number Field
Fieldnames(1) = "TESTPROFIELD3"    ' return User defined Field
oWorkItemsX.Criteria.CaseFieldNames = Fieldnames ' replaces existing array
                                           ' of fieldnames
' Enable WorkItemsX to be sorted by Casenumber

Debug.Print "<=== Show list of preset/default sortfields_
            (ie see std client) ===>"
cnt = UBound(oWorkItemsX.Criteria.SortFields)
For i = LBound(oWorkItemsX.Criteria.SortFields) To cnt
    SortFlds = oWorkItemsX.Criteria.SortFields
    Set oSortField = SortFlds(i)
    Debug.Print ("Sortfield Field Name = " & oSortField.FieldName)
Next

' Create and configure Sortfield
Set oSortField = New SWSortField
oSortField.FieldName = "SW_CASENUM"
oSortField.IsAscending = False
oSortField.SortAsType = swNumericSort
' Add SortField
ReDim SortFldArray(0)
Set SortFldArray(0) = oSortField
oWorkItemsX.Criteria.SortFields = SortFldArray
Debug.Print "<=== List sortfields after configuring criteria on Xlist ===>"
cnt = UBound(oWorkItemsX.Criteria.SortFields)
For i = LBound(oWorkItemsX.Criteria.SortFields) To cnt
    SortFlds = oWorkItemsX.Criteria.SortFields
    Set oSortField = SortFlds(i)
    Debug.Print ("Sortfield Field Name = " & oSortField.FieldName)
Next

```

See [page 339](#) for a comprehensive example.

Note for TIBCO iProcess Objects (C++) Only - On SWCriteriaWI and SWCriteriaC, SWSortField objects are copied. Once the array of SWSortFields is set (i.e., call setSortFields), you need to delete each of the SWSortField objects (i.e., delete pSortField).

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 effected 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.

System Fields used in Sorting

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 TIBCO iProcess Engine through the system fields is also available to the client through properties on SWWorkItem and SWCase. For example, **SW_CASENUM** is available to the client in the **SWCase.CaseNumber** property. The TIBCO iProcess Engine, however, doesn’t have access to those properties, so the property names from SWWorkItem and SWCase can’t be used in filter and sort expressions — instead, the system field names need to be used in your expressions.

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 the TIBCO iProcess Objects Server or the database). (See the section [“Can the WIS Perform the Sort Operation?”](#) on [page 107](#) and [page 134](#) 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 expired flag (1 - expired; 0 - all others)	SW_EXPIRED	Numeric	1	X	X	
Deadline set flag (1 - has deadline; 0 - all others)	SW_HASDEADLINE	Numeric	1	X	X	
Forwardable work item flag (1 - forwardable; 0 - all others)	SW_FWDABLE	Numeric	1	X	X	

Sort Criteria	System Field	Data Type	Length	WIS-compatible	Applies to work items	Applies to cases
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 ^b	X	
Mail ID	SW_MAILID	String or Numeric ^c	7 (integer) 45 (string)		X	
Outstanding work items in case (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	7	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 others)	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	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	
Terminated date and time of the case	SW_TERMINATED	DateTime	16			X
Unopened work item (1 - unopened; 0 - all others)	SW_NEW	Numeric	1	X	X	
Urgent flag (1- urgent; 0 - all others)	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. SW_LOCKER is WIS-compatible only if your TIBCO iProcess Objects Server has implemented CR 13397.
- c. 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 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 sort 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 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 as a sort criteria. These fields have been superseded by CDQP fields (see above) as they were considered too limiting since there are only four of them.

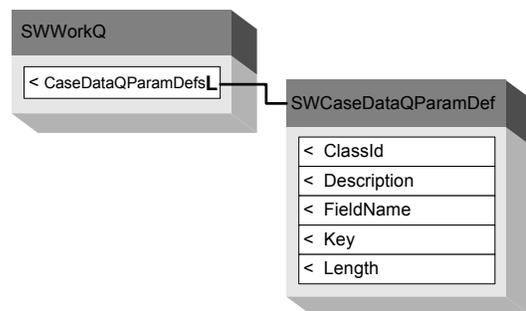
More about CDQP and work queue parameter fields are described in the following subsections.

Using Case Data Queue Parameter Fields

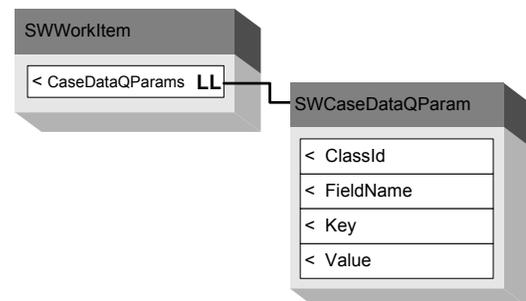
Case Data Queue Parameter (CDQP) fields provide an efficient method of sorting on the value of fields in your work items. To make use of this functionality, you must first pre-designate the fields you want to sort on as CDQP fields. Fields are designated as CDQP fields with the utility, **swutil**. This utility is used to create a list, on the TIBCO iProcess Engine, of the case data fields that are available to use for sorting. See the *TIBCO iProcess Engine Administrator's Guide* for information about using **swutil**.

Note - Case Data Queue Parameter fields are also used for efficiently filtering on case data, as described in the Filtering Work Items and Cases chapters.

Once you have created the list of CDQP fields with **swutil**, this list of fields is available in the **SWWorkQ.CaseDataQParamDefs** property. This property contains a list of **SWCaseDataQParamDef** objects, one for each case data field that has been designated as a CDQP field in the work queue. This list tells you the CDQPs that are available for filter and sort criteria.



The **CaseDataQParams** property on **SWWorkItem** provides access to CDQPs that are being used in the work item. Note, however, to control resource usage, you can specify which CDQPs to return from the server by using the **CDQPNames** property on the **SWCriteriaWI** object. By default, all CDQPs used in a work item are returned from the server. The **SWCaseDataQParam** object contains the current value in the CDQP.



After being designated as a CDQP field, that field can be used to sort on. (In the example below, assume LAST_NAME has been designated as a CDQP):

```
Dim oSortField As New SWSortField
.
.
oSortField.FieldName = LAST_NAME
oView.SortFields.Add oSortField
```

CDQPs Contain Work Item Data

If the WIS is performing the sort operation, and you are using CDQP fields in your 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 sort operation, and you are using CDQP fields in your 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 readme to determine if CR 12425 is implemented in your server.)

See “[Case Data vs. Work Item Data](#)” on page 91 for more information about the difference between Work Item Data and Case Data.

Using Work Queue Parameter Fields

Note - Previous versions of TIBCO iProcess Objects 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 (CDQP)” fields has been implemented (see the previous section). New development should use CDQP 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 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 *Filtering Work Items and Cases* chapters.)

If you have case/field data that 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 assign the Work Queue Parameter field to an SWSortField object, then add that object to the SortFields property. For example:

```
Dim oSortField As New SWSortField
.
.
oSortField.FieldName = SW_QPARAM1
oView.SortFields.Add oSortField
```

This method of sorting on case data fields is fast and efficient — it's performed by the Work Item Server. If you directly added the case data field to the SortFields property, the sort would have to be performed by the TIBCO iProcess Objects Server.

The **SWWorkItem** object has four read-only properties that provide access to the values in the Work Queue Parameter fields — they are **WQParam1 - WQParam4**. These properties will contain the values you place in fields, SW_QPARAM1 - SW_QPARAM4, for each work item.

The **SWWorkQ** object has four read-only properties that contain a name for each of the Work Queue Parameter fields (WQParam1-4). If you use the TIBCO iProcess Client, 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 Client (Windows) Manager's Guide*.

Setting Default Sort Criteria

You can set *default* sort criteria for a work queue that persists on the queue. This causes future SWViews or SWXLists of work items on the current instance of the queue to use this default criteria.

Note - If you use the TIBCO iProcess Client, sort criteria that are defined on the Work Queue Manager Work Queue Sort Criteria dialog become the default sort criteria for that work queue. When an SWView or SWXList object is created for that work queue, the sort criteria defined on that dialog are written to the SortFields property.

The following methods on the **SWWorkQ** object 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):

- **SetDefCriteria** - This method sets the default filter and sort criteria for this work queue. It uses the current setting of the **FilterExpression** property and the **SortFields** property on the **SWView** in the **WorkItems** property in the current instance of the work queue to establish the default criteria.

Note that since this method uses the filter and sort criteria in the view in the WorkItems property, this method is practical to use if you are using SWViews, but not if you are using SWXLists. If you are using SWXLists, the SetDefCriteriaEx method is a better choice (see below).

- **SetDefCriteriaEx** - This method allows you to specify the default filter and sort criteria for this work queue by passing in the criteria as parameters. This causes the criteria you pass in this

method to persist on this instance of the work queue, causing future SWViews or SWXLists of work items on this instance of the queue to use this default criteria.

- **ClearDefCriteria** - This method clears the default sort criteria that were set either through the Work Queue Manager or by using the **SetDefCriteria** or **SetDefCriteriaEx** methods (see above). (This also clears any default filter criteria that have been defined.)

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 call SetDefCriteria/SetDefCriteriaEx.

The following are the sort criteria that are supported by the Work Queue Manager, that can, therefore, be persisted with the SetDefCriteria/SetDefCriteriaEx methods:

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
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

Implied Sort Fields for Multiple Views/XLists

If you create multiple views of cases (with **MakeViewCases**) or work items (with **MakeViewItems**), there is an implied sort field on which the view is sorted before any user-defined sort criteria are applied.

- Multi views of **cases** are always sorted first by **procedure number**. This is because the view created with the **MakeViewCases** method can possibly span multiple procedures and/or nodes.
- Multi views of **work items** are always sorted first by **queue name**. This is because the view created with the **MakeViewItems** method can possibly span multiple work queues.

Any user-defined sort criteria are applied after the view is sorted on the implied primary sort field.

Sorting as a Specified Data Type

The **SortAsType** property on **SWSortField** allows you to specify that the sort fields be converted to the specified data type before the sort comparison is performed.

Sorting by a specified type with the **SortAsType** property is only applicable when sorting on:

- Work Queue Parameter fields (SW_QPARAM1-4) - see [“Using Work Queue Parameter Fields” on page 185](#)
- Case Data Queue Parameter fields - see [“Using Case Data Queue Parameter Fields” on page 184](#)
- Case Description (SW_CASEDESC)

The data types that can be specified with the **SortAsType** property are enumerated in **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.

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

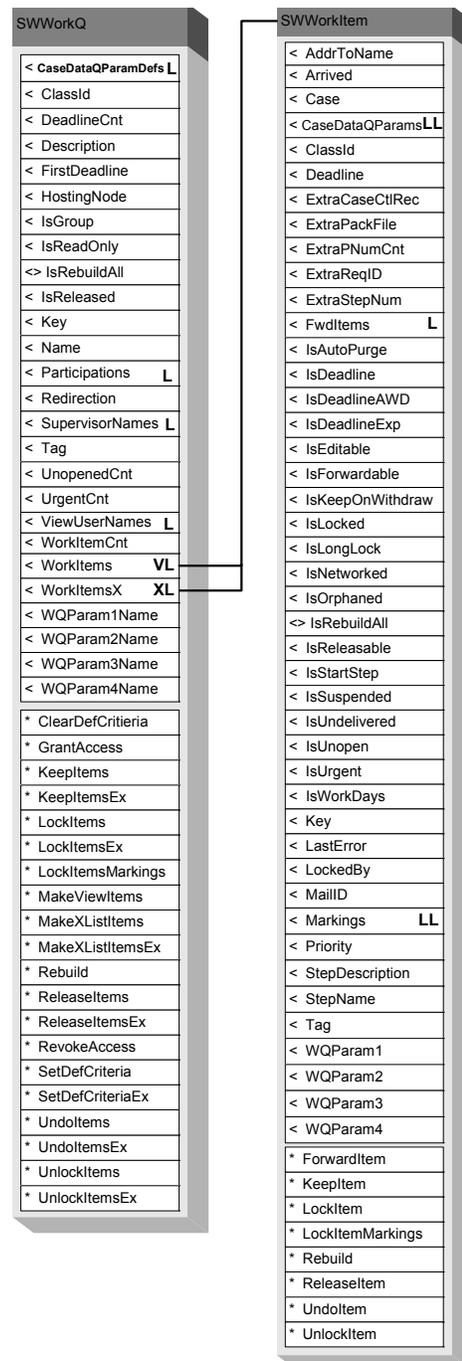
Work queues are represented by the **SWWorkQ** object. A list of SWWorkQ objects is available from two locations:

- **SWNode.WorkQs** - This property returns a list of SWWorkQ objects, one for each work queue defined on the node. This includes both released and test work queues (note that a test work queue will exist only if there are work items in it — explained below) .
- **SWUser.WorkQs** - This property returns a list of SWWorkQ objects, one for each work queue the user is authorized to work on. This includes only released work queues.

The SWWorkQ object contains properties that allow you to view information about the work queue (e.g., whether or not it's released (IsReleased), the number of work items in the work queue (WorkItemCnt), etc.). It also contains methods that allow you to act upon the work items in the work queue (e.g., locking work items (LockItems), releasing work items (ReleaseItems), etc.). These functions are described in later subsections of this chapter.

Work Item Objects

A work item represents an individual item in a work queue. Work items are represented in SSO by **SWWorkItem** objects. These objects provide access to information about that specific work item, e.g., whether or not the work item is locked (IsLocked), the deadline for the work item (Deadline), etc.



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 with the **SWNode.WorkQs** property (only released work queues are returned in **SWUser.WorkQs**). 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.

The “key” (Key property) for the SWWorkQ object contains the test/released designation as follows:

```
WorkQName@HostingName|Mode
```

where Mode = T for test or R for released.

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 “[Procedure Status](#)” on page 43 for information about the procedure statuses available) 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 “[Participation Access to a Work Queue](#)” on page 204).

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 accessing the **IsReleased** property on SWWorkQ (you can access SWWorkQ only after a work item has been sent to the work queue). (The **IsWorkQReleased** property is available on SWOutstandingItem to determine if the outstanding item is located in a test or released work queue.)

Accessing Work Items on a Work Queue

The properties and methods that you use to access the work items on a work queue depends on whether you are using views (SWViews) or XLists (SWXLists) in your client application. Accessing work items in each of these list types is described in the following subsections.

Note - It is highly recommended that you use SWXLists instead of SWViews to list work items because of their improved efficiency — see “[Handling Large Lists of Work Items, Cases, Users, OS Users, Groups](#)” on page 275 for more information.

Accessing Work Items in SWViews

You can obtain two types of views of work items in a work queue:

- The “default” view - The **WorkItems** property on **SWWorkQ** returns a view (SWView) of all of the work items in the work queue that satisfy the filter criteria in the **SWView.FilterExpression** property. They are listed in the view in the order specified by the **SWView.SortFields** property. This is the view of work items you will typically use.

```
Dim oWorkItems As SWView
Set oWorkQ = oUser.WorkQs.ItemByKey(cboWorkQs.Text)
Set oWorkItems = oWorkQ.WorkItems

' Read down the Work Queue, actioning each Work Item
For Each oLocalItem In oWorkItems
    .
    .
```

- An “alternate” view - If you have a need to show a view of work items in more than one way, additional views can be created with the **MakeViewItems** method. For example, you may want to show a view of work items sorted by priority and also (concurrently) a view of unopened work items that arrived today. To do this, you would create an additional SWView containing SWWorkItem objects filtered and sorted in the desired way.

The **MakeViewItems** method is available from several objects, which governs its scope, as shown in the table below.

Method	Called From	Scope
MakeViewItems	SWWorkQ	Used to view work items in the work queue.
	SWUser	Used to view work items across multiple work queues on one node.
	SWEntUser	Used to view work items across multiple work queues on more than one node.

If you create a view with the **MakeViewItems** method, the **Status** property on that view will always return **swChanged**.

The “single-parameter” **SWNode.MakeViewItemsByTag** method is available to create an additional view of work items by passing a single *Tag* parameter. This method would typically be used by a web-based application. Its scope is the work queues on the node from which it is called. (See “[Stateless Programming](#)” on page 270 for more information about using “single-parameter” methods in a web-based environment.)

Note that the SWViews returned with the **WorkItems** property and **MakeViewItems** method are not automatically populated with SWWorkItem objects. See “[How SWViews are Created and Populated at the Client](#)” on page 66 for details.

Accessing Work Items in SWXLists

You can obtain two types of XLists of work items in a work queue:

- The “default” XList - The **WorkItemsX** property on **SWWorkQ** returns an XList (SWXList) of all of the work items in the work queue that satisfy the filter criteria in the **SWCriteriaWI.FilterExpression** property. They are listed in the XList in the order specified by the **SWCriteriaWI.SortFields** property. This is the XList of work items you will typically use.

```
Dim oWorkItems As SWXList
Set oWorkQ = oUser.WorkQs.Item(lboWorkQs.ListIndex)
Set oWorkItems = oWorkQ.WorkItemsX
    .
    .
```

- An “alternate” XList - If you have a need to show an XList of work items in more than one way, additional SWXLists can be created with the **MakeXListItems** or **MakeXListItemsEx** methods on **SWWorkQ**. For example, you may want to show an XList of work items sorted by priority and also (concurrently) an XList of unopened work items that arrived today. To do this, you would create an additional SWXList containing SWWorkItem objects filtered and sorted in the desired way. (Note that this is the method you need to use to create an XList of work items that you are going to persist — see [“Working with Persisted XLists” on page 81](#) for more information.)

Note that the SWXList returned by the **WorkItemsX** property and the **MakeXListItems / MakeXListItemsEx** methods are not automatically populated with SWWorkItem Objects. See [“Populating an XList of Work Items” on page 77](#) for details.

Determining the Number of Work Items in a Work Queue

The method you use to determine the number of work items in a work queue depends on whether you are using SWViews or SWXLists, as described in the following subsections.

Determining the Number of Work Items in a Work Queue on an SWView

You can determine the total number of work items available from the server to an SWView at the client by looping through the view and checking the **SWView.IsEOL** property. When **IsEOL** returns True, it means “the last item you accessed is the last item available to the list.” At this point, the **Count** property indicates the total number of work items available from the server. See the example below.

```
i = 0
oWorkQList.IsEOL = False
While (oWorkQList.IsEOL = False)
    Debug.Print "Name: " & oWorkQList(i).Name & "; Description: " _
        & oWorkQList(i).Description
    Debug.Print "oWorkQList.Count = " & oWorkQList.Count
    i = i + 1
Wend
```

It’s important to understand that the **IsEOL** property is **ONLY** true when the last item you accessed was the last item in the list. If another item in the list is subsequently accessed, **IsEOL** is then set to false.

Other count properties are also available on SWView:

- **ExcludeCnt** - Contains the number of work items that were not included in the indexed collection at the server because they did not satisfy the criteria in the FilterExpression property. (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)
- **InvalidCnt** - Contains the number of work items not included in the indexed collection because they were invalid within the context of the filter criteria (e.g., the filter expression references a field name not defined in all work items). (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)

Determining the Number of Work Items in a Work Queue on an XList

You can determine the number of work items in a work queue when you are using SWXLists by using the following properties:

- **ItemCount** - The total number of items at the server the first time the XList was accessed or the last time it was rebuilt. This count includes only those work items that satisfy the filter criteria in **SWCriteriaWI.FilterExpression**.

This count is available immediately upon creating the XList (unlike on a view where you need to loop through the list until IsEOL is true to determine the total number of items available on the server).

```
If oWorkItemsX.ItemCount > 25 Then
    cnt = 25                ' if more than 25 only loop through 1st 25
Else
    cnt = oWorkItemsX.ItemCount ' if less than 25 loop through all of them
End If
```

- **Count** - The number of items that are currently stored in the XList at the client.
- **ExcludeCnt** - Contains the number of work items that were not included in the indexed collection at the server because they did not satisfy the criteria in the FilterExpression property. (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)
- **InvalidCnt** - Contains the number of work items not included in the indexed collection because they were invalid within the context of the filter criteria (e.g., the filter expression references a field name not defined in all work items). (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)

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.
- **Keeping** a work item causes changes that have been made to the 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 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 process 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. Several methods are provided to lock a work item:

- **LockItem** and **LockItemMarkings** - These methods on **SWWorkItem** are used to lock a single work item. These are used if your work items are in an **SWView**. The difference: **LockItem** returns either the visible markings on the form or all markings (both visible and conditional) on the form; **LockItemMarkings** allows you to specify which markings to return — see the next subsection.
- **LockItems** - This method on **SWWorkQ** is used to lock multiple work items in the queue. This is used if your work items are in an **SWView**.
- **LockItemsEx** and **LockItemsMarkings** - These methods on **SWWorkQ** are used to lock one or more work items in the queue. These are used if your work items are in an **SWXList**. The difference: **LockItemsEx** returns all markings (both visible and conditional) on the form; **LockItemsMarkings** allows you to specify which markings to return — see the next subsection.

Getting Markings When Locking Work Items

Locking work items also causes the **Markings** property to become populated with **SWMarking** objects. This allows you access to the data in those markings (**SWMarking.Value**). Note, however, that the **Markings** property is populated differently depending on which method is used to lock the work items, as follows:

- **LockItem** – The *allMarkings* parameter allows you to specify that either all markings on the form (both visible and conditional), or only the visible markings, be returned.
- **LockItemMarkings** – The *MarkingNames* parameter is used to specify the specific markings that are to be returned, allowing you to control resource usage.
- **LockItems** – All markings on the form (both visible and conditional) of each work item are returned.
- **LockItemsEx** – All markings on the form of each work item are returned.
- **LockItemsMarkings** – The *MarkingNames* parameter is used to specify the specific markings that are to be returned, allowing you to control resource usage.

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 Client). If the work item is locked in this way, the **SWWorkItem.IsLocked** property will be 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 from TIBCO iProcess Objects using one of the methods described above (LockItem, etc.). If the work item is locked in this way, the **SWWorkItem.IsLongLock** property will be 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 **IsLongLock** 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 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 Objects and you attempt to lock it in the Work Queue Manager.

Unlocking a Work Item

A work item is automatically unlocked when it is “kept” or “released” (described later in this chapter).

The following methods allow you to unlock a long-locked (locked via TIBCO iProcess Objects) work item:

- **UnlockItem** - This method on **SWWorkItem** is used to unlock a single work item. This is used on **SWViews** of work items.
- **UnlockItems** - This method on **SWWorkQ** is used to unlock multiple work items. This is used on **SWViews** of work items.
- **UnlockItemsEx** - This method on **SWWorkQ** is used to unlock one or more work items. This is used on **SWXLists** of work items.

The unlock methods cause all changes that have been made to the work item since the last “keep” to be discarded and unlocks the work item (the **IsLongLock** property is set to False).

Any user can unlock a work item they have locked, but you must have system administrator authority (MENUAME = ADMIN) to unlock a work item that another user has locked. That is the primary purpose of the unlock methods — to allow a System Administrator to unlock another user's work items. (See “[User Attributes](#)” on page 221 for information about the MENUAME attribute.)

Work items that have been locked in the Work Queue Manager (**IsLocked** is True) cannot be unlocked using TIBCO iProcess Objects. (For information about unlocking work items using the Work Queue Manager, see the *TIBCO iProcess Client (Windows) Manager's Guide*.)

Discarding Changes made to a Locked Work Item

The following methods are available to discard changes that have been made to a long-locked (locked via TIBCO iProcess Objects) work item:

- **UndoItem** - This method on **SWWorkItem** is used to discard changes to a single work item. This is used on **SWViews** of work items.
- **UndoItems** - This method on **SWWorkQ** is used to discard changes to multiple work items. This is used on **SWViews** of work items.

- **UndoItemEx** - This method on **SWWorkQ** is used to discard changes to one or more work items. This is used on **SWXLists** of work items.

The undo methods cause all changes that have been made to the work item since the last “keep” to be discarded and unlocks the work item (the **IsLongLock** property is set to False).

Note that you can also discard any changes since the last “keep” *without* unlocking the work item by calling the **LockItem** method again. This essentially re-populates the **Markings** property, overwriting any new data in the markings.

Has a Work Item been Locked/Opened?

When a work item is added to a work queue, its **IsUnopen** property 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 TIBCO iProcess Objects), then keep it in the same work queue, its **IsUnopen** flag is changed to False, indicating that it has been opened and worked on (but not necessarily modified).

The **SWWorkQ** object also contains an **UnopenedCnt** property, which indicates 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

The name of the user who currently has a work item locked is available in the **SWWorkItem.LockedBy** property.

Executing a Command when a Work Item is Locked

When a procedure is defined with the TIBCO iProcess Modeler 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 will appear in the **SWCommand.InitialExpr** property. This 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 following methods are available to keep work items:

- **KeepItem** - This method on **SWWorkItem** is used to keep a single work item. This is used if your work items are in an **SWView**.
- **KeepItems** - This method on **SWWorkQ** is used to keep multiple work items in the queue. This is used if your work items are in an **SWView**.
- **KeepItemsEx** - This method on **SWWorkQ** is used to keep one or more work items in the queue. This is used if your work items are in an **SWXList**.

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 91](#) for more information.

When you modify data in a marking of a work item, the **IsSendValue** flag is automatically set to True. This flag tells the system to send the new marking data to the Work Item Data storage area when the work item is kept. You can optionally set the **IsSendValue** flag back to False after changing marking data if you decide you do not want that data sent to the Work Item Data storage area — this essentially discards the change made to the marking when you keep the work item.

Executing a Command when a Work Item is Kept

When a procedure is defined with the TIBCO iProcess Modeler 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 will appear in the **SWCommand.KeepExpr** property. This 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 the work queue and the process to advance to the next step in the procedure. The following methods are available to release work items:

- **ReleaseItem** - This method on **SWWorkItem** is used to release a single work item. This is used if your work items are in an **SWView**.
- **ReleaseItems** - This method on **SWWorkQ** is used to release multiple work items in the queue. This is used if your work items are in an **SWView**.
- **ReleaseItemsEx** - This method on **SWWorkQ** is used to release one or more work items in the queue. This is used if your work items are in an **SWXList**.

When you release a work item, data associated with that work item 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 91](#) for more information.

When you modify data in a marking of a work item, the **IsSendValue** flag is automatically set to True. This flag tells the system to send the new marking data to the Case Data storage area when the work item is released. You can optionally set the IsSendValue flag back to False after changing marking data if you decide you do not want that data sent to the Case Data storage area — this essentially discards the change made to the marking when you release the work item.

Validating Markings

All of the release item methods provide a *Validate* parameter that allows you to ensure that all required markings are sent to the server upon release.

- If *Validate* = **True**: 1) Validate that the markings exist on the form, 2) validate that all required markings (**swRequired**) on the form are sent to the server with non-empty data, and 3) validate that display markings (**swDisplay**) are *not* sent to the server. The **IsSendValue** flag for each marking is set to False by default when the work item is locked. IsSendValue is automatically set to True if the value of the marking is changed. You can also force the sending of the marking by setting IsSendValue to True before releasing the work item.
- If *Validate* = **False** (the default), it bypasses the enforcement of marking types on the Staffware form.

Executing a Command when a Work Item is Released

When a procedure is defined with the TIBCO iProcess Modeler 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 will appear in the **SWCommand.ReleaseExpr** property. This 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 **StartCaseEx** method provides a *Release* 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 *Release* 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 231](#) for more information.

What is an Orphaned Work Item?

When a work item is released, the **SWWorkItem.IsOrphaned** property on that work item is set to True. This flag tells you that the work item has been released, even though it may still appear in the work queue. This is most apt to occur with group queues where a user on one machine has released the work item and a user on another machine attempts to access that work item. The work item will be removed from the work queue the next time the queue is cleared and rebuilt.

Determining if a Work Item could not be Delivered to the Addressee

When a work item is released, the process advances to the next step in the case. The work item’s “addressee” specifies to whom the next work item is to be sent. If the work item cannot be delivered to that addressee, the **SWWorkItem.IsUndelivered** property is set to True and the work item is placed in the **\$undeliv** work queue. This could occur if the user or group specified as the addressee no longer exists, or the addressee is a field that evaluated to a user or group that doesn’t exist. (Only the **swadmin** user has access to the **\$undeliv** work queue.)

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 **SWWorkItem.IsReleasable** property is set to True.

Errors Resulting from Processing Work Items

When locking, unlocking, keeping, releasing, or undoing multiple work items on the **SWWorkQ** object, the operation could possibly fail for one or more of the work items in the queue. Rather than totally backing out of the operation, an error code is written to the **SWWorkItem.LastError** property for the property or properties that failed.

It is up to the client to look at LastError to determine if the operation was successful or not. If the operation was successful, LastError will contain a 0 (zero).

See the on-line help system for a list of the possible error codes.

Forwarding Work Items to Another Work Queue

You can forward work items to another user's or group's work queue in the following ways:

- **Manually Forwarding** - This allows you to forward one work item to a specified work queue.
- **Auto Forwarding** - This allows you to set up a schedule so work items are automatically forwarded to another work queue during a specified time period.

These are described in detail in the following subsections.

Manually Forwarding Work Items

The **SWWorkItem** object contains a **ForwardItem** method that allows you to forward the work item represented by the **SWWorkItem** object to a specified work queue. The following factors determine whether or not you can forward a work item:

- **Work Queue Access** - To call the **ForwardItem** method, you must have access to the work queue containing the work item you want to forward.
- **The step's Forward Permission** - This is specified when the step is defined in the TIBCO iProcess Modeler. 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** property (which is available in the step definition, **SWStep**, as well as the work item objects, **SWWorkItem** and **SWOutstandingItem**).

If the step's forward permission is set (**IsForwardable** = True), the **FwdItems** property on **SWWorkItem** is populated with a list of **SWFwdItem** objects, one for each work queue to which the work item can be forwarded. Note that the list of work queues in the **FwdItems** property is not totally definitive; even though it's populated, you may not be able to forward the work item, depending on the user's forward permission. Conversely, if **FwdItems** is NOT populated, you may still be able to forward the work item, depending on the user's forward permission.

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 the 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 **ForwardItem** that is used. Internally, the system logs in as the user who owns the work queue from which the work item is being forwarded — 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 221 for information about modifying the value of the USER-FLAGS attribute.

To manually forward a work item, call the **ForwardItem** method and pass the name of the destination node and work queue.

Auto Forwarding/Redirecting Work Items

TIBCO iProcess Objects originally provided the following methods to allow you to automatically forward work items to another user’s or group’s work queue:

- **CreateAutoFwd** - This method is used to create an “auto forward” record, which defines where work items are to be auto forwarded.
- **DeleteAutoFwd** - This method cancels a previously created auto forward record.
- **AutoFwds** - This property returns a list of **SWAutoFwd** objects, which identify the work items to auto forward to another work queue.

These methods have been superseded by a new way of forwarding work items, which is now called “redirection”. These older methods are still supported. Note, however, that on Windows systems, there is still a process (**SWEntObjDB.exe**) that runs to support the older method of forwarding work items. You can disable this process, if desired, by doing the following (note that it is disabled by default starting with version 10.5.0):

1. Create the following Registry string:

```
\HKEY_LOCAL_MACHINE\SOFTWARE\Staffware plc\Staffware EntObj Server\Nodes\  
NodeName\DB_Enabled
```

where *NodeName* is the name of your TIBCO iProcess Objects Server.

2. Set **DB_Enabled** to 0 (zero) to disable the SWEntObjDB.exe process.

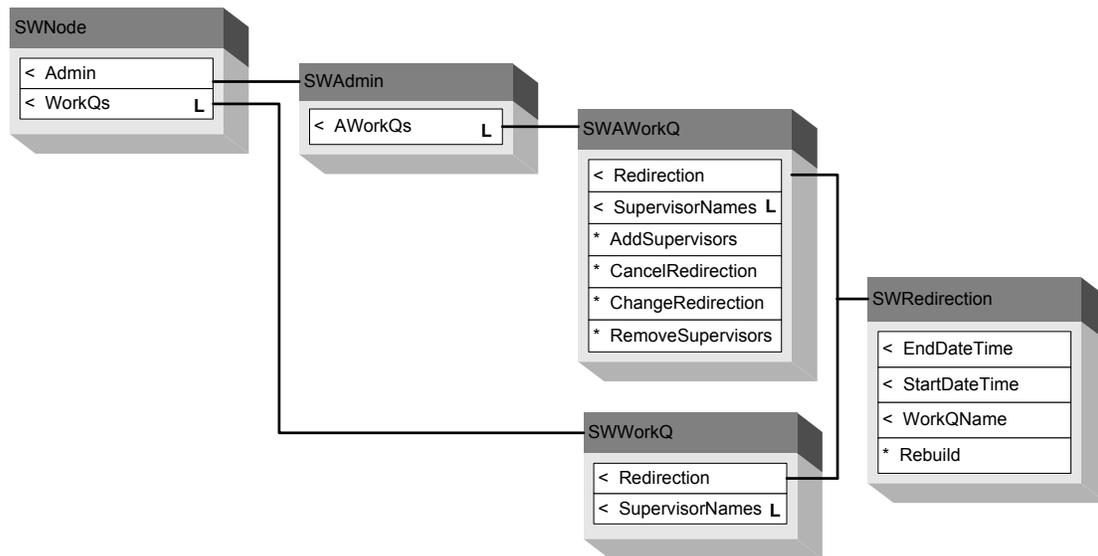
Note - This also disables “view-only work queue access”. See “[View-Only Access to a Work Queue](#)” on page 203.

The following subsections describe how to forward work items to another work queue using redirection.

Redirection of Work Items

Redirection functionality allows you to specify that all work items destined for a work queue be redirected to another work queue 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 the work queue to which the work items are to be redirected, as well as the date and time the redirection is to start and end.

The illustration below shows the objects, properties, and methods involved in defining redirection.



Each work queue on the node has a corresponding “administrative” work queue (an **SWWorkQ** object). Each administrative work queue is used to administer its corresponding runtime work queue (**SWWorkQ** object), including setting up a redirection schedule. (The administrative work queue is used to administer other functionality also, such as participation, which is described later in this chapter. Note that the **SWWorkQ** object contains more properties and methods than are shown above; this illustration shows only those that apply to redirection.)

Notice that the work queue’s redirection schedule (**SWRedirection**) is available in the **Redirection** property on both the administrative work queue (**SWWorkQ**) and runtime work queue (**SWWorkQ**). However, it can be modified only from the administrative work queue.

*Note - The step’s forward permission (defined in the step definition with the TIBCO iProcess Modeler) and the user’s forward permission (defined in the USERFLAGS attribute) are not used by redirection. They are only used when forwarding individual items with the **ForwardItem** method.)*

Creating a Redirection Schedule

A redirection schedule is defined by the **SWRedirection** object. This object contains the following redirection elements:

- Name of the user or user group to whom the work items are being redirected (`WorkQName`)
- Date and time to start the redirection (`StartDateTime`)
- Date and time to end the redirection (`EndDateTime`)

When the **SWWorkQ** object is created, an **SWRedirection** object is automatically created with empty values. The **ChangeRedirection** method is used to modify that initial redirection schedule. (There isn’t a “CreateRedirection” method — one is automatically created — you just need to change it to fit your needs.)

The **ChangeRedirection** method replaces the existing **SWRedirection** object in the **Redirection** property on both the **SWWorkQ** and **SWWorkQ** objects. To cancel an existing redirection schedule, use the **CancelRedirection** method. Canceling a schedule changes the schedule’s elements back to empty values.

To create a redirection schedule, a user must be designated as a supervisor of the work queue. See [“Work Queue Supervisors” on page 208](#).

If you make a change to or cancel an existing redirection schedule with the `ChangeRedirection` method, call **Rebuild** on the `SWRedirection` object to make the new schedule active (rather than rebuilding the entire work queue).

Only one redirection schedule can be defined for a work queue.

Using the SWDate Object (Java and C++ Clients Only)

The **SWDate** object is used in redirection (and participation) schedules to hold the start and end date/time. This object was created to be able to specify an empty date/time, which can't be done with the java **Date** object. The **SWDate** object is only used in Java and C++ clients — it is not part of the COM object model.

Java Clients

The **SWDate** object has two public constructors that allow you to either pass in a java **Date** object that is used as either the start or end date/time, or to set the object to “Empty” (True) or “NotEmpty” (False):

```
public SWDate(Date jDate)

public SWDate(boolean setEmpty)
```

Due to the requirements of redirection, it is necessary to have the **SWDate** object represent any of the following:

- **DateTime** (Year, Month, Day, Hour, Minute, Seconds) - This is used when constructing an **SWDate** object that will actually hold a date/time to be used as the *StartDateTime* or *EndDateTime* argument in the redirection schedule.
- **Empty** - This causes an empty string to be sent to the server. This takes on different meanings depending on which parameter it is representing, as follows:
 - *StartDateTime* - Causes the redirection to start immediately.
 - *EndDateTime* - Causes redirection to last indefinitely.
- **NotEmpty** - Causes the previously assigned date/time to be used.

An example of how you might create an **SWDate** object using these formats is shown below:

```
Calendar oCal = new GregorianCalendar();

//Setting up a start date for redirection
oCal.clear();

// Set date/time to Sept 12, 2002, 4:30 PM
oCal.set(2002, 8, 12, 16, 30, 0);
SWDate StartDateTime = new SWDate(oCal.getTime());

// Set end date/time to infinite
SWDate EndDateTime = new SWDate(true);
```

C++ Clients

This section describes using the **SWDate** object to specify the date and time to start and end the redirection when using TIBCO iProcess Objects (C++).

The following method is provided on the **SWDate** object to specify this information:

- **setDateTime** - Sets the date/time in the **SWDate** object.

After setting the date/time with the **setDateTime** method, the **SWDate** object can be passed as a parameter with the **changeRedirection** method to specify either a start date/time or an end date/time. Setting the date/time with this method also causes the **SWDate.isExplicit** flag to be set to true.

Methods are also provided on the **SWDate** object to specify that you want to use the default value, or that you want to use what is currently defined in the redirection schedule:

- **setDefault** - Specifies that the default for the start date/time or end date/time be used. The meaning of the default is context specific. Calling this method also causes the **SWDate.isDefault** flag to be set to true.
- **setNoChange** - Specifies that the date/time value currently defined in the redirection schedule should be used. Calling this method also causes the **SWDate.isNoChange** flag to be set to true.

Granting Access to a Work Queue

By default, a user has access to his own work queue, as well as group work queues for which the user is a member. Access to a work queue can also be granted to other users as follows:

- **View-only access** - This gives another user the ability to view work items in a work queue, without the ability to make changes to the work items.
- **Participation access** - This gives another user full access to the work items in a work queue.

View-Only Access to a Work Queue

The following methods and properties provide read-only access functionality:

- **SWWorkQ.GrantAccess** - This method allows you to specify one or more users who will now have view-only access to the work items in this work queue. You must have system administrator authority (MENUNAME = ADMIN) to call this method. (See [“User Attributes” on page 221](#) for information about the MENUNAME attribute.)
- **SWWorkQ.RevokeAccess** - This method revokes the view-only access that was previously granted with the **GrantAccess** method. You must have system administrator authority (MENUNAME = ADMIN) to call this method. (See [“User Attributes” on page 221](#) for information about the MENUNAME attribute.)
- **SWWorkQ.ViewUsersName** - This property tells you who currently has view-only access to this work queue.
- **SWUser.ViewOnlyQs** - This property tells you which work queues this user has been granted view-only access with the **GrantAccess** method.

These older methods of granting access are still supported. Note, however, that on Windows systems, there is still a process (**SWEntObjDB.exe**) that runs to support this older view-only access. You can disable this process, if desired, by doing the following (note that it is disabled by default starting with version 10.5.0):

1. Create the following Registry string:

```
\HKEY_LOCAL_MACHINE\SOFTWARE\Staffware plc\Staffware EntObj Server\Nodes\  
NodeName\DB_Enabled
```

where *NodeName* is the name of your TIBCO iProcess Objects Server.

2. Set **DB_Enabled** to 0 (zero) to disable the SWEntObjDB.exe process.

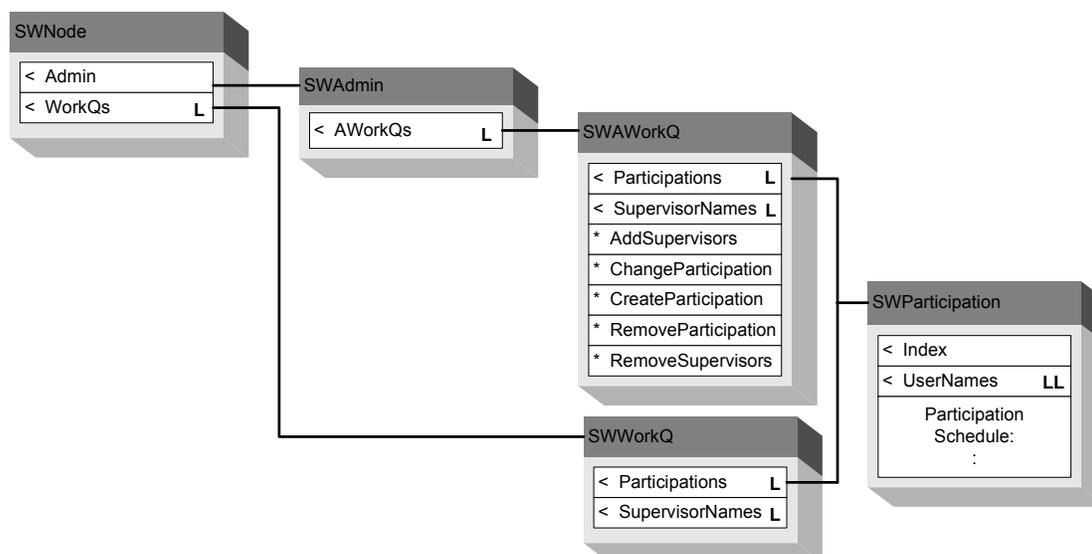
Note - This also disables the older method of auto-forwarding work items. See “Auto Forwarding/Redirecting Work Items” on page 200.

Participation Access to a Work Queue

“Participation” allows you to specify that a user can “participate” in the work queue of another user, that is, they have full read/write 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.

The illustration below shows the objects, properties, and methods involved in defining participation of a work queue.



Each work queue on the node has a corresponding “administrative” work queue (an **SWAWorkQ** object). Each administrative work queue is used to administer its corresponding runtime work queue (**SWWorkQ** object), including setting up a participation schedule. (The administrative work queue is used to administer other functionality also, such as redirection, which is described earlier in this chapter. Note that the **SWAWorkQ** object contains more properties and methods than are shown above; this illustration shows only those that apply to participation.)

Notice that the work queue’s participation schedules (**SWParticipation**) are available in the **Participations** property on both the administrative work queue (**SWAWorkQ**) and runtime work queue (**SWWorkQ**). However, they can be created and modified only from the administrative work queue.

*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.*

Creating a Participation Schedule

A participation schedule is defined by the **SWParticipation** 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

SWParticipation objects are created by calling the **CreateParticipation** method on SWWorkQ. This method adds the new SWParticipation object to the **Participations** list on both the SWWorkQ and SWAWorkQ objects. After adding an SWParticipation object to the Participations list, you must rebuild the list for the new participation schedule to take effect.

Existing participation schedules (i.e., SWParticipation objects) can be modified or removed from the Participations list by using the **ChangeParticipation** and **RemoveParticipation** methods, respectively. Again, you must rebuild the list for these changes to be reflected in the list.

To create a participation schedule, a user must be designated as a supervisor of the work queue. See [“Work Queue Supervisors” on page 208](#).

Using the SWDate Object (Java and C++ Clients Only)

The **SWDate** object is used in participation (and redirection) schedules to hold the start and end date/time. This object was created to be able to specify an empty date or time, which can't be done with the java **Date** object. The SWDate object is only used in Java and C++ clients — it is not part of the COM object model.

Java Clients

The **SWDate** object has two public constructors that allow you to either pass in a java **Date** object that is used as either the start or end date/time, or to set the object to “Empty” (True) or “NotEmpty” (False):

```
public SWDate(Date jDate)

public SWDate(boolean setEmpty)
```

Due to the requirements of participation, it is necessary to have the **SWDate** object represent any of the following:

- **DateOnly** (Year, Month, Day) - This is used when constructing an **SWDate** object that will actually hold a start or end date to be used as the *StartDate* or *EndDate* argument in the participation schedule.

- **TimeOnly** (Hour, Minute, Seconds) - This is used when constructing an **SWDate** object that will actually hold a start or end time to be used as the *StartTime* or *EndTime* argument in the participation schedule.
- **Empty** - This causes an empty string to be sent to the server. This takes on different meanings depending on which parameter it is representing, as follows:
 - *StartDate* - Causes participation to begin on the next date allowed by the *IsSunday-IsSaturday* parameters.
 - *EndDate* - Causes participation to last indefinitely.
 - *StartTime* - Causes participation to start directly after midnight on the days that participation is allowed (according to the other parameters).
 - *EndTime* - Causes participation to end at midnight on the days that participation is allowed (according to the other parameters).
- **NotEmpty** - Causes the previously assigned date or time or be used.

An example of how you might create an **SWDate** object to be used in a participation schedule is shown below:

```
Calendar oCal = new GregorianCalendar();

// Setting up a date
oCal.clear();
// Set date to May 31, 2002
oCal.set(2002, 4, 31);
SWDate StartDate = new SWDate(oCal.getTime());

// Setting up a time
oCal.clear();
// Set time to 4:30 PM
oCal.set(0, 0, 0, 16, 30, 0);
SWDate StartTime = new SWDate(oCal.getTime());
// Testing Date for equality
SWDate oDate = oParticipation.getStartDate();

if (oDate.dateEquals(StartDate))
    .
    .

// Testing Time for equality
SWDate oTime = oParticipation.getStartTime();

if (oTime.timeEquals(StartTime))
    .
    .
```

C++ Clients

This section describes using the **SWDate** object to specify the date and time to start and end the participation when using TIBCO iProcess Objects (C++).

The following methods are provided on the SWDate object to specify this information:

- **setDate** - Sets the date in the SWDate object.
- **setTime** - Sets the time in the SWDate object.

After setting either the date or time with one of the methods above, the SWDate object can be passed as a parameter with the **createParticipation** or **changeParticipation** method to specify either a start date, start time, end date, or end time. Setting the date or time with one of these methods also causes the **SWDate.isExplicit** flag to be set to true.

Methods are also provided on the SWDate object to specify that you want to use the default value, or that you want to use what is currently defined in the participation schedule:

- **setDefault** - Specifies that the default for the start date, end date, start time, or end time be used. The meaning of the default is context specific. Calling this method also causes the **SWDate.isDefault** flag to be set to true.
- **setNoChange** - Specifies that the date or time value currently defined in the participation schedule should be used. Calling this method also causes the **SWDate.isNoChange** flag to be set to true.

The TIBCO iProcess Objects Server Maintains an Index of the Participation Schedules

The TIBCO iProcess Engine does not provide an index or a key to identify a particular participation schedule in the Participations list. To get around this, a session is started each time the Participations property is accessed on the administrative work queue object (SWAWorkQ). The sole purpose of this session is to allow the TIBCO iProcess Objects Server to maintain an index for the list of participation schedules. This session will remain open until the Participations list is destroyed (i.e., the SWAWorkQ object is destroyed).

There are a couple of issues you need to be aware of because of these sessions being opened to impose indexes on the Participations list:

- As the number of sessions that are open increases, the performance of the TIBCO iProcess Objects Server is negatively affected. Therefore, it is important that you are aware of when sessions are opened and that they are closed as soon as possible (by destroying the SWAWorkQ object).

Note that a session is opened only when the Participations property is accessed on the SWAWorkQ (administrative) object, not on the SWWorkQ (runtime) object. Therefore, if you only need to view the participation schedules, do so from SWWorkQ. Only use the Participations property on SWAWorkQ to determine the index values of the schedules for the purpose of using the index with the ChangeParticipation or RemoveParticipation methods.

- Conflicts may occur when changing a participation schedule. Opening a session on a Participations list does not lock the list. It only causes the TIBCO iProcess Objects Server to maintain an index on the list while the session is open. Therefore, while the client is administering a specific participation schedule, it is possible that another client, or even someone using the Process Client, could be administering the same schedule. The first to finish will modify the participation schedule at the TIBCO iProcess Engine. When the second one attempts to modify the same

schedule, the TIBCO iProcess Engine will not recognize it, causing the TIBCO iProcess Objects Server to generate an error.

The index that is created for each participation schedule is maintained in the **SWParticipation.Index** property. This index value is needed with the **ChangeParticipation** and **RemoveParticipation** methods to specify which schedule you want to change or remove. You will need to step through the Participations list to determine which schedule you want to modify or remove, then obtain the schedule's index value from the **Index** property.

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 by calling the **SupervisorNames** property from either the “runtime” work queue object, **SWWorkQ**, or the “administrative” work queue object, **SWAWorkQ**.

Note that the same list of user names returned by the **SupervisorNames** property can be obtained from the **QSUPERVISOR** user attribute. See [“User Attributes” on page 221](#) for information about **QSUPERVISOR**.

From the standpoint of a user, you can also determine which work queues the user has been authorized to supervise by calling the **SWUser.AdminQNames** property. This property returns a list of strings, one for each work queue for which the user is a work queue supervisor.

Adding Work Queue Supervisors

To add a new supervisor to a work queue, use the following method:

- **SWAWorkQ.AddSupervisors** - This method allows you to specify that one or more users is a supervisor for the work queue. The caller of the **AddSupervisors** method must have system administrator authority (**MENUNAME = ADMIN**). (See [“User Attributes” on page 221](#) for information about the **MENUNAME** attribute.)

After calling **AddSupervisors**, you must rebuild the list to have the changes reflected in the **SupervisorNames** list.

Removing Work Queue Supervisors

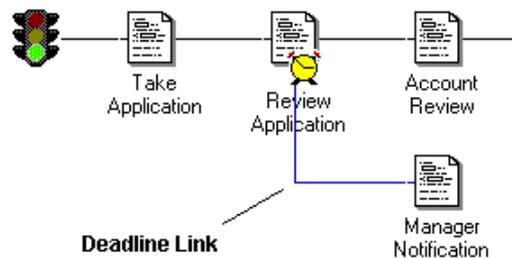
To remove a supervisor from a work queue, use the following method:

- **SWWorkQ.RemoveSupervisors** - This method allows you to remove one or more users from the list of supervisors for the work queue. The caller of the **RemoveSupervisors** method must have system administrator authority (**MENUNAME = ADMIN**). (See [“User Attributes” on page 221](#) for information about the **MENUNAME** attribute.)

After calling **RemoveSupervisors**, you must rebuild the list to have the changes reflected in the **SupervisorNames** list.

Work Item Deadlines

When a step is defined using the TIBCO iProcess Modeler, 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 **SWStep.Deadline** property returns an **SWDeadline** object, which contains deadline dates, times, criteria, etc.

If the work item has a deadline defined, the **SWWorkItem.IsDeadline** property is set to True.

Once a work item is in a work queue, the following properties are available on **SWWorkItem** to provide information about the deadline on that work item:

- **Deadline** - The date and time the deadline expires. This property will contain 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.
- **IsDeadlineExp** - Boolean value that indicates whether or not the deadline has expired.

The **SWWorkQ** object also contains a **FirstDeadline** property that returns the date and time of the earliest deadline in the work queue.

Withdrawing Work Item on Deadline

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 the TIBCO iProcess Modeler, 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 Application step, the process flows to the Manager Notification step, and the work item for the Review Application is withdrawn from the work queue.

The deadline withdrawal option can be determined by accessing the following property on the **SWWorkItem** 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.

Deadline Counts

You can determine the number of work items that have deadlines by using the following properties:

- **SWWorkQ.DeadlineCnt** - This tells you the number of work items in the queue that have a deadline.
- **SWCriteriaWI.DeadlineCnt** - This tells you the number of work items in the **SWXList** that have a deadline.

Filtering and Sorting on Deadline Information

The following system fields are also available for filtering and sorting work items on deadline information:

- **SW_HASDEADLINE** - Deadline set flag (1 - has deadline, 0 - all work items)¹
- **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)¹

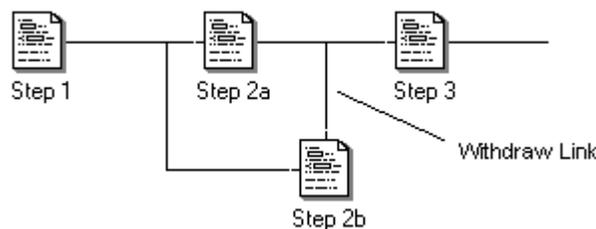
See the appropriate *Filtering Work Items and Cases* chapter and the *Sorting Work Items and Cases* chapter for information about how to use these system fields when filtering and sorting.

1. When filtering on system fields that can be set to 0 or 1, they work in this 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_EXPIRED** is set to 1, this means “display only the work items that have expired deadlines”. If it’s set to 0, this means “don’t display only the work items with expired deadlines, instead, display all of them.”

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 “[Withdrawing Work Item on Deadline](#)” on page 209.
- A “withdraw link” can be drawn when the step is defined in the TIBCO iProcess Modeler.



In the example shown above, if Step 2a is released before Step 2b, then Step 2b is automatically withdrawn from the work queue.

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 the TIBCO iProcess Modeler, check the **Don't delete work items on withdraw** box. You can determine whether or not this option has been checked by accessing the **IsKeepOnWithdrawal** property on the step definition object (SWStep), or the **SWorkItem** 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 in the **OutstandingItems** list SWCase).
- 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 step), 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.

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 a third-party application that is external to the client application. An example is an EAI step, which causes the TIBCO iProcess Engine to pass field data, a 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 work item back to the client when it is finished.

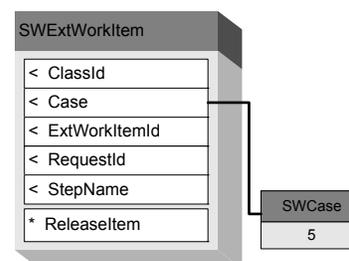
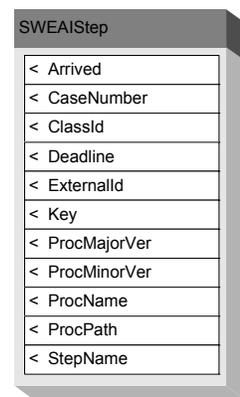
When the process flow reaches an EAI step, an **SWEAISTep** object is generated that represents the outstanding EAI step. This object can be accessed with the **EAISteps** property on **SWCase**. The external ID generated by the engine, and which is passed to the third-party application, is available in the **ExternalId** property on **SWEAISTep**.

The form definition that is passed to the third-party application is available in the **ExtForm** property on the **SWStep** object (currently only applicable to EAI steps).

When an EAI step becomes outstanding, it also generates an **SWExtWorkItem** object, representing the external work item. This object is available with the **GetExtWorkItem** method on **SWNode**. Calling the **GetExtWorkItem** method requires that you pass the external ID that uniquely identifies the external work item.

The **SWExtWorkItem** object contains the following properties and methods:

- **Case** - This property returns an **SWCase** object representing the case in which the external work item resides.
- **ExtWorkItemId** - This property returns the unique ID that was passed from the client application to the third-party application to identify the external work item.
- **RequestId** - This property returns an identifier for the external work item (this is an identifier that is available for all outstanding work items — it's not the one used by the external application).
- **StepName** - This property returns the name of the step that generated the external work item.
- **ReleaseItem** - This method is used to release the external work item once the external application is finished with it. See the *Releasing an External Work Item* section below for more information.



Releasing an External Work Item

The third-party application releases an outstanding EAI step using the **ReleaseItem** method on the **SWExtWorkItem** object. This method requires that the external ID that was passed to the third-party application be passed as a parameter in the **ReleaseItem** method to identify the specific outstanding EAI step (there could be multiple EAI steps outstanding at one time).

When the `ReleaseItem` 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 *NextStep* parameter. If you specify an alternative next step with the *NextStep* parameter, you can also use the *DoActions* parameter to specify how the process flow should advance from the EAI step, as follows:

- If the *DoActions* 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 *NextStep* parameter.
- If the *DoActions* parameter is `False` (the default), the process only advances to the step specified in the *NextStep* 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.

13

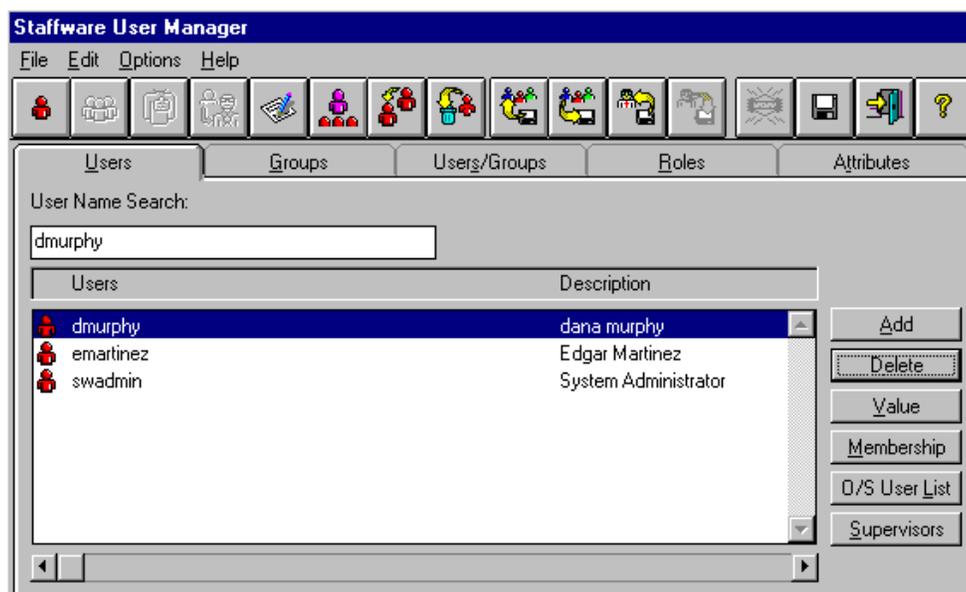
User Administration

Introduction

User administration tasks can be performed in two ways:

- using the Staffware User Manager, or
- through properties and methods on objects.

The **Staffware User Manager** is part of the Administration Managers suite of utilities. It is used to administer users, groups, roles, and attributes, which are all associated with Staffware users.



Note - The term “Staffware User Manager” is a remnant of the original software created by Staffware, which was purchased by TIBCO. This dialog (as well as the dialogs for the other Administration Managers — Staffware List Manager, Staffware Table Manager, etc.) still contain the “Staffware” name. This name will be used here until those dialogs are changed.

All of the user administration functions that can be performed using the Staffware User Manager can also be done using TIBCO iProcess Objects. For information about using the Staffware User Manager, see the *TIBCO iProcess Client (Windows) Manager's Guide*.

The remainder of this chapter describes performing user administration tasks by using the TIBCO iProcess Objects.

Types of Users

TIBCO iProcess 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 a Staffware 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. For more information, see [“Is an O/S User needed for every Staffware User?” on page 217](#).
- **Staffware User** - This is a user created for the purpose of logging into a client application. A Staffware user is created using the **CreateUser** method on the **SWNode** object.
- **Enterprise User** - This user is created for the purpose of representing a user across all nodes in the enterprise. The username used by the enterprise user does not necessarily correspond with a Staffware user username. This is because a Staffware user may be known by one username on Node1, another username on Node2, and still another username on Node3. The enterprise user (**SWEntUser** object) allows that user to log into multiple nodes, but be known by one username across all of them. See [“Creating Enterprise Users” on page 31](#).

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** configuration parameter to 1 (UNIX), or check the appropriate box on the *TIBCO iProcess Objects Server Configuration Utility Users* tab (Windows). See [“ImplicitMoveSysInfo” on page 323](#) for more information.

- **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** configuration parameter to 0 (UNIX), or uncheck the appropriate box on the *TIBCO iProcess Objects Server Configuration Utility Users* tab (Windows). See [“ImplicitMoveSysInfo” on page 323](#) for more information. The actual **MOVESYSINFO** function can then be executed by calling the **MoveSysInfo** method on **SWNode**. (You must have system administrator authority (MENUNAME = ADMIN) to call the **MoveSysInfo** method. See [“User Attributes” on page 221](#) for information about the MENUNAME attribute.) See your on-line help for information about this method.

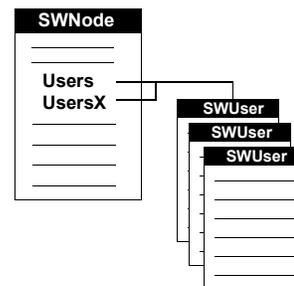
- **Using swutil** - If changes to a large number of users, groups, roles, attributes, or queue supervisor definitions need to be made, your best option might be to use the **swutil** “Update User Information” function:

```
swutil USERINFO filename
```

For more information, see the *TIBCO iProcess Engine Administrators Guide*.

Staffware Users

Each node maintains a list of Staffware users that have been created on that node. This list of users is maintained in the **Users** property (as an SWList) and the **UsersX** property (as an SWXList). Each user is represented by one **SWUser** object. (See [“Handling Large Lists of Work Items, Cases, Users, OS Users, Groups”](#) on page 275 for information about why you would use UsersX instead of Users.)



Note - The term “Staffware user” is a remnant of the original software created by Staffware, which was purchased by TIBCO. This term is still used in the TIBCO iProcess Engine dialogs and documentation, and will be used here until the engine dialogs are changed.

Remember that if you add or delete a user from a node, you will need to rebuild the SWList or SWX-List before the change is reflected in the Users/UsersX properties.

Creating a Staffware User

A Staffware user is created on a specific node with the **SWNode.CreateUser** method. The following is an example:

```
Dim sAttributes(0), sValues(0)

If txtNewUser <> "" Then
    If txtUserDescription.Text <> "" Then
        sAttributes(0) = "Description": sValues(0) = txtUserDescription.Text
        Call oNode.CreateUser(txtNewUser.Text, sAttributes, sValues)
    Else
        oNode.CreateUser (txtNewUser.Text)
    End If
    txtNewUser.Text = "": txtUserDescription.Text = ""
End If
Exit Sub
```

The CreateUser method allows you to optionally specify groups to add the user to, and attribute values to assign to the user (as shown in the example above). (You must have system administrator authority (MENUNAME = ADMIN) to call the CreateUser method. See [“User Attributes”](#) on page 221 for information about the MENUNAME attribute.) 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.

Note - The Staffware user being created may have to already exist as an OS user, depending how your system is configured. See [“Is an O/S User needed for every Staffware User?”](#) on page 217.

When a Staffware user is created, a corresponding directory for that user is added 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 a “test” and “released” work queue for the user to be created. See [“Test vs. Released Work Queues”](#) on page 190 for more information.

Deleting a Staffware User

One or more Staffware users can be deleted from the node with the **SWNode.DeleteUsers** method. (You must have system administrator authority (MENUNAME = ADMIN) to call the DeleteUsers method. See “[User Attributes](#)” on page 221 for information about the MENUNAME attribute.) An example is shown below.

```
If lboUsers.Text <> "" Then
    oNode.DeleteUsers (lboUsers.Text)
End If
Exit Sub
```

When a Staffware 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 the user is not the addressee of a step, because after the user is deleted, their work queue is no longer accessible.

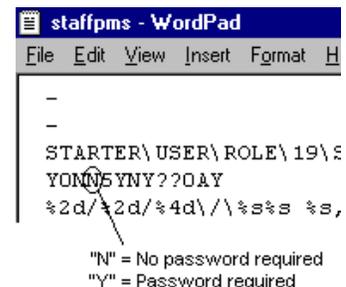
Is an O/S User needed for every Staffware User?

You can configure your system so that when you create a Staffware user with the **CreateUser** method, it does not require that the user already exists as an O/S user. To specify that an O/S user is NOT required for each new Staffware user, the following two parameters must be configured:

- Password checking on the TIBCO iProcess Engine must be turned off.

The *\$\$SWDIR/etc/staffpms* (UNIX) and *SWDIR/etc/staffpms* (Windows) file can be modified to enable or disable O/S password checking. This is done by specifying a “Y” or “N” in the 4th character of the 4th line in the **staffpms** file.

- The TIBCO iProcess Objects Server **CheckOSUser** configuration parameter must be set to 0 (zero). See [page 322](#) for information about setting this configuration parameter.



Note - If you are creating the user with the Staffware User Manager, the user must exist in the operating system.

Changing the User’s Password

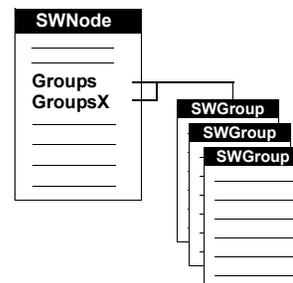
A Staffware user’s password can be changed with the **SWUser.ChangePassword** method.

Staffware 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 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.

A group is specific to the node on which it is created. Each node maintains a list of its groups in the **Groups** property (as an SWList) and the **GroupsX** property (as an SWXList). Each group is represented by one **SWGGroup** object. (See [“Handling Large Lists of Work Items, Cases, Users, OS Users, Groups” on page 275](#) for information about why you would use GroupsX instead of Groups.)



Remember that if you add or delete a group from a node, you will need to rebuild the SWList or SWXList before the change is reflected in the Groups/GroupsX properties.

The **SWUser** object also has a **Groups** property that contains a list of all of the groups to which that specific user belongs.

Creating a User Group

A group is created on a specific node with the **SWNode.CreateGroup** method. (You must have system administrator authority (MENUNAME = ADMIN) to call the CreateGroup method. See [“User Attributes” on page 221](#) for information about the MENUNAME attribute.) The following is an example:

```
If txtNewGroup <> "" Then
    oNode.CreateGroup (txtNewGroup.Text)
End If
```

The CreateGroup method also 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 group is created, a corresponding directory for that group is added at *SWDIR\queues\group-name* (Windows) or *\$\$WDIR/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 a “test” and “released” work queue for the group to be created. See [“Test vs. Released Work Queues” on page 190](#) for more information.

Deleting a User Group

One or more groups can be deleted from the node with the **SWNode.DeleteGroups** method. (You must have system administrator authority (MENUNAME = ADMIN) to call the DeleteGroups method. See “[User Attributes](#)” on page 221 for information about the MENUNAME attribute.) An example is shown below.

```
If lboGroups.Text <> "" Then
    oNode.DeleteGroups (lboGroups.Text)
End If
```

When a group is deleted, its corresponding directory at *SWDIR\queues\groupname* (Windows) or *SSWDIR/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 the group is not the addressee of a step, because after the group is deleted, their work queue is no longer accessible

Adding and Removing Users to/from a Group

Once an SWGroup object is created with the CreateGroup method (or through the User Manager), you can add and remove users from the group with the **SWGroup.AddUsers** and **SWGroup.RemoveUsers** methods, respectively. (You must have system administrator authority (MENUNAME = ADMIN) to call these methods. See “[User Attributes](#)” on page 221 for information about the MENUNAME attribute.) An example of adding users to a group is shown below.

```
If lboGroups.Text <> "" And cboUser.Text <> "" Then
    Set oGroup = oNode.Groups.ItemByKey(lboGroups.Text)
    oGroup.AddUsers (cboUser.Text)
End If
```

Roles

A role is a job title or function, such as Account Manager. One Staffware 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 specific to the node on which they are created. Each node maintains an SWList of its roles in the **Roles** property. Each role is represented by one **SWRole** object. Remember that if you add or delete a role from a node, you will need to rebuild the SWList before the change is reflected in the Roles property.

The **SWUser** object also has a **RoleNames** property that contains a list of all of the roles to which that specific user belongs.

Creating a Role

A role is created on a specific node with the **SWNode.CreateRole** method. (You must have system administrator authority (MENUNAME = ADMIN) to call the CreateRole method. See [“User Attributes” on page 221](#) for information about the MENUNAME attribute.) The following is an example:

```
If txtRoleName <> "" And cboRoleUser <> "" Then
    oNode.CreateRole txtRoleName.Text, cboRoleUser.Text
    txtRoleName.Text = "": cboRoleUser.Text = ""
End If
```

Deleting a Role

One or more roles can be deleted from the node with the **SWNode.DeleteRoles** method. (You must have system administrator authority (MENUNAME = ADMIN) to call the DeleteRoles method. See [“User Attributes” on page 221](#) for information about the MENUNAME attribute.) An example is shown below.

```
If lboRoles.Text <> "" Then
    oNode.DeleteRoles (lboRoles.Text)
End If
```

User Attributes

User attributes are properties/characteristics of a Staffware user or group. There are two types of user attributes:

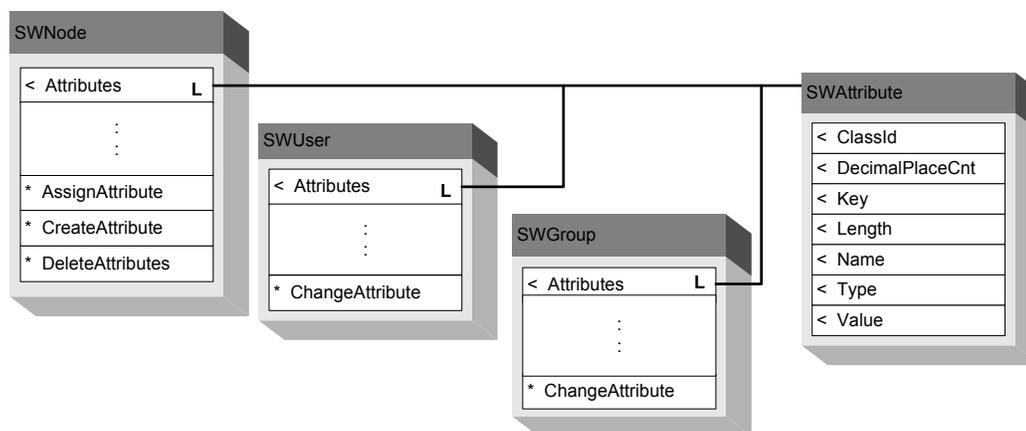
- **Customizable** - You can create attributes that can hold any type of characteristic of the user/group. Some examples are an employee number, purchasing authority, etc.
- **Pre-defined** - Every Staffware user and group that is created is assigned each of the six pre-defined attributes are shown in the table below.

Attribute	Description
DESCRIPTION	If the user or group was created with the CreateUser or CreateGroup method, this defaults to the name of the user or group. If the user or group was created with the Staffware 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	<p>This specifies the user's access authority (the name is derived from "which menus the user has access to"). This attribute is only applicable to users (not groups). The possible values are:</p> <ul style="list-style-type: none"> • 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 the TIBCO iProcess Modeler for the purpose of defining procedures. • ADMIN - This is the system administrator authority. Users with this authority can perform administrative-type functions. See "User Authority" on page 227 for a list of the functions a system administrator can perform. <p>The user's MENUNAME attribute is also available in the SWUser.MenuName and SWActiveUser.Menu properties.</p> <p><i>Note - You cannot change the value of this attribute for the swadmin user.</i></p>
QSUPERVISOR	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 in the SupervisorNames property. See " Work Queue Supervisors " on page 208 for more information.

Attribute	Description
SORTMAIL	<p>Specifies the sequence in which work items are sorted in the user's or group's work queue. The possible values are:</p> <ul style="list-style-type: none"> • 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	<p>Specifies the user's ability to forward work items to another work queue. The possible values are:</p> <ul style="list-style-type: none"> • " " - (Empty string) Work items from this user's work queue can be forwarded if the step's Forward permission has been set in the procedure definition (on the Step Status dialog). 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 cannot be forwarded from this user's work queue, even if the step's Forward permission is set. (This is called Forward None in the User Manager.) <p>See "Manually Forwarding Work Items" on page 199 for more information.</p>

Attributes are defined on a specific node. 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.

The **Attributes** property on **SWNode**, **SWUser**, and **SWGroup** contains an **SWList** of **SWAttribute** objects, one for each attribute defined on the node.



The **AssignAttribute** method on SWNode allows you to assign a value to an attribute for one or more users or groups. The **ChangeAttribute** method on SWUser and SWGroup allows you to change the value of an attribute for that particular user or group.

Creating an Attribute

An attribute is created on a specific node with the **SWNode.CreateAttribute** method. (You must have system administrator authority (MENUAME = ADMIN) to call the CreateAttribute method.)

```
Dim AttributeType As SWAttributeType
If txtAddAttribute.Text = "" Then
   RetVal = MsgBox("Enter an Attribute Name.", vbExclamation)
    txtAddAttribute.SetFocus
Else
    If Not IsNumeric(txtLength.Text) Then
       RetVal = MsgBox("Enter a numeric value for length.", vbExclamation)
        txtLength.SetFocus
    ElseIf Not IsNumeric(txtDecimal.Text) Then
       RetVal = MsgBox("Enter a numeric value for decimal.", vbExclamation)
        txtDecimal.SetFocus
    Else
        Screen.MousePointer = vbHourglass
        'determine data type of attribute
        If optText.Value Then
            AttributeType = swTextAttr
        ElseIf optNumeric.Value Then
            AttributeType = swNumericAttr
        ElseIf optDate.Value Then
            AttributeType = swDateAttr
        ElseIf optTime.Value Then
            AttributeType = swTimeAttr
        End If
        'create new attribute
        oNode.CreateAttribute txtAddAttribute.Text, AttributeType, _ Val(txt-
            Length.Text), txtDefault.Text, Val(txtDecimal.Text)
        Screen.MousePointer = 0
        Unload frmAddAttribute
    End If
End If
Exit Sub
```

When an attribute is added to the node, it is automatically assigned to each existing user and group on that node.

Deleting an Attribute

One or more attributes can be deleted from the node with the **SWNode.DeleteAttributes** method. (You must have system administrator authority (MENUAME = ADMIN) to call the DeleteAttributes method.)

```
Set oNode = oEntUser.LoggedInNodes.Item(cboAdminServers.ListIndex)
If lboSWAttributes.ListIndex >= 0 Then
    RetVal = MsgBox("Are you sure you want to remove " & lboSWAttributes.Text_
        & "?", vbYesNo)
    If RetVal = vbYes Then
        Screen.MousePointer = vbHourglass
        oNode.DeleteAttributes lboSWAttributes.Text
        Call RefreshAdminList(lboSWAttributes, oNode.Attributes)
        Screen.MousePointer = 0
    End If
Else
    RetVal = MsgBox("Select an Attribute.", vbExclamation)
End If
```

Caution - The system will allow you to delete the pre-defined attributes that it needs to function properly.

Modifying an Attribute

The **ChangeAttribute** method on **SWUser** and **SWGGroup** allows you to change the value of a specific attribute for the user or group. (You must have system administrator authority (MENUAME = ADMIN) to call the ChangeAttribute method.)

```
oUser.ChangeAttribute oAttribute.Name, txtValue.Text
```

Make sure the data type of the new value you provide in the ChangeAttribute method matches the data type for the attribute you are changing.

Why isn't the new User, Group, Role or Attribute Appearing in the List?

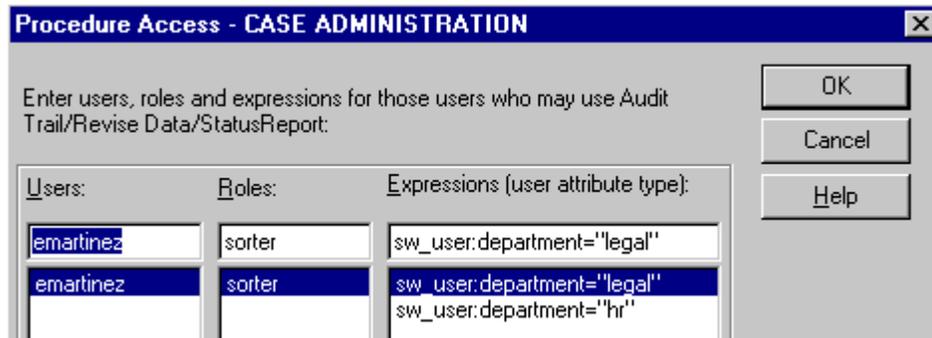
Staffware users, groups, roles, and attributes are created by the background process, so there is a delay between when the method is executed and when the new user, group, role, or attribute actually appears in the list (Users, UsersX, Groups, etc.).

Remember that you must also Rebuild the list to see any newly added (or deleted) items in the list.

Determining which Procedures a User can Audit

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 the **Procedure | Access | Case Admin** function in the TIBCO iProcess Modeler:



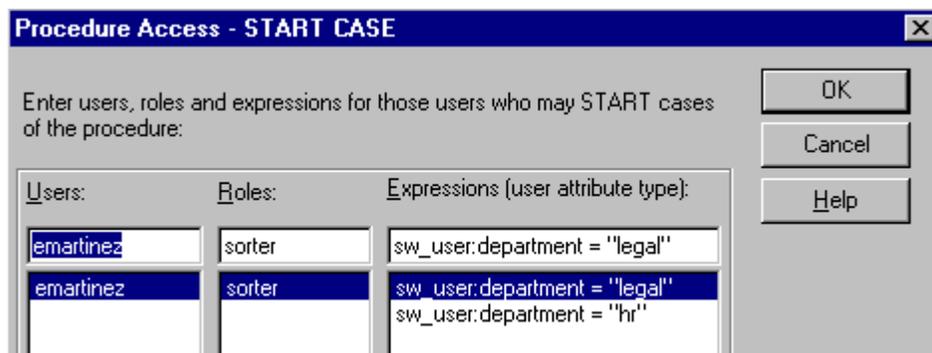
This dialog is used to define who has **case administration** authority. If a user, group, or role name is specified using this dialog, the ability to audit this procedure is limited to *only* the users, groups, or roles specified, or the users for which the expression(s) evaluates to True. Note that 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 **swadmin** user always has access to the audit trail of any procedure.) If no one is designated as having access through this function, it defaults to giving *everyone* access.

You can determine which procedures a user can audit by using the **AuditProcs** property on **SWUser**. This property returns an SWList of **SWProc** objects, one for each procedure the user has permission to audit. See the example below.

```
'Display in the lboProcs list box the procedures on this node
'that the logged-in-user can audit
'
With lboProcs
  .Clear
  For Each oProc In oNode.LoggedInUser.AuditProcs
    lboProcs.AddItem (oProc.Name)
  Next
End With
```

Determining the Procedures for which the User 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. You can, however, specify the users who can start a case of a procedure by using the **Procedure | Access | Case Start** function in the TIBCO iProcess Modeler.



Note - A procedure must be “released” (SWProc.Status = “swReleased”) before users other than the procedure owner and swadmin can start cases of that procedure.

If a user name or role is specified using this function in the TIBCO iProcess Modeler, case-start access to this procedure is limited to only the users, groups, or roles specified, or 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.

You can determine the procedures for which a user can start a case by using the **StartProcs** property on the **SWUser** object.

```
'Get the procedures logged-in-user can start cases of
'and display in list box lboStartProcs
'
Set oUser = oNode.LoggedInUser
lblUser.Caption = oUser.Name

lboStartProcs.Clear
For Each oProc In oUser.StartProcs
    lboStartProcs.AddItem (oProc.Name)
Next
Exit Sub
```

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 **ChangeAttribute** method on **SWUser**. See [“User Attributes” on page 221](#) for more information.

*Note - To ensure that there is always a user that has system administrator authority, there is a special system administrator user (**swadmin**) whose MENUNAME attribute cannot be changed from ADMIN.*

- **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 the TIBCO iProcess Modeler). Note that by default, when a procedure is defined in the TIBCO iProcess Modeler, *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 which Procedures a User can Audit” on page 225](#) 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 CloseByCriteria
Purge cases	PurgeCases PurgeByCriteria PurgeAndReset
Create/delete users	CreateUser DeleteUsers
Create/delete groups	CreateGroup DeleteGroups
Modify group membership	AddUsers RemoveUsers

Function	Methods
Create/delete/modify attributes	CreateAttribute DeleteAttributes AssignAttribute ChangeAttribute
Create/delete roles	CreateRole DeleteRoles
Unlock work item locked by another user <i>(Note - Any user can unlock a work item that they have locked.)</i>	UnlockItem UnlockItems UnlockItemsEx
Modify auto-forward records for other users <i>(Note - Any user can modify an auto-forward record for themselves.)</i>	CreateAutoFwd DeleteAutoFwd
Add/remove queue supervisors	AddSupervisors RemoveSupervisors
Change TIBCO iProcess Objects Server log settings	ResetLog SetLogCategories SetLogLevel SetLogTrace SetMaxLogSize
Add/delete view-only queue access	GrantAccess RevokeAccess
Forward work items from other user's work queues <i>(Note - For information about the permission requirements to forward work items from your own work queue, see page 199.)</i>	ForwardItem
Move system information (MOVESYSINFO function)	MoveSysInfo

- You must have **case administration** authority to perform the following functions:

Function	Methods
Retrieve audit data for cases of the procedure	SWCase.AuditSteps
Rebuild a list audit steps	SWCase.AuditSteps.Rebuild

- 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	Cases CasesX
Retrieve case data for any procedure	SWCase.Fields
Get the filtered case count for any procedure	GetFilteredCaseCnt
Make a case (stateless) of any procedure on the node	MakeCase MakeCaseByTag
Rebuild a list of cases or fields for any procedure	SWCase.Rebuild SWCase.Fields.Rebuild

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 **StartCaseEx** method on the SWProc object. This method takes the form:

```
StartCaseEx([Description], [StartStepName], [Release], [Validate], [FieldNames], _
[FieldValues], [SubProcPrecedence])
```

Note - The StartCase method on SWProc has been deprecated — it will be removed in a later release of the software. All new code needs to use StartCaseEx.

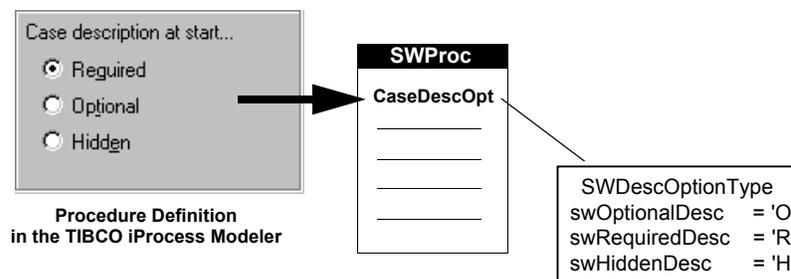
By default, the case starts on the first step defined in the procedure — the name of this step is stored in the **SWProc.StartStepName** property. However, you can optionally cause the procedure to start on a step other than the one specified in the procedure definition — this is done by providing the *StartStepName* parameter when calling the StartCaseEx method.

*Note - You cannot directly start a case (with StartCaseEx) of a procedure that is defined as a sub-procedure (if it is a sub-procedure, its **IsSubProc** property will be set to True). Sub-procedures can only be started from a sub-procedure call step, dynamic sub-procedure call step, or graft step.*

Case Description

The procedure definition (defined with the TIBCO iProcess Modeler) specifies whether or not the StartCaseEx method requires a *Description* parameter.

- If the procedure definition specifies that the description is **required** (SWProc.CaseDescOpt = swRequiredDesc), the *Description* parameter must be provided with the StartCaseEx method.
- If the procedure definition specifies that the description is **optional** or **hidden** (SWProc.CaseDescOpt = swOptionalDesc or swHiddenDesc), the *Description* parameter does not need to be provided with the StartCaseEx method.

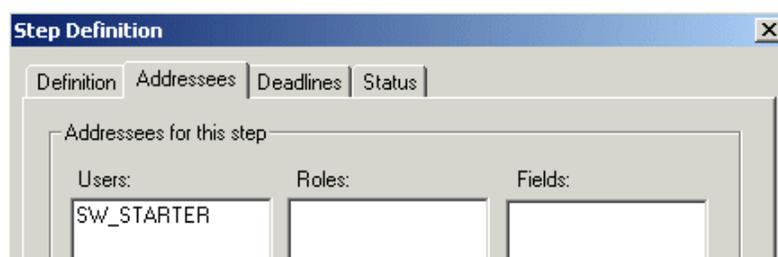


Keeping/Releasing the Start Step

The `StartCaseEx` method provides a *Release* 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.

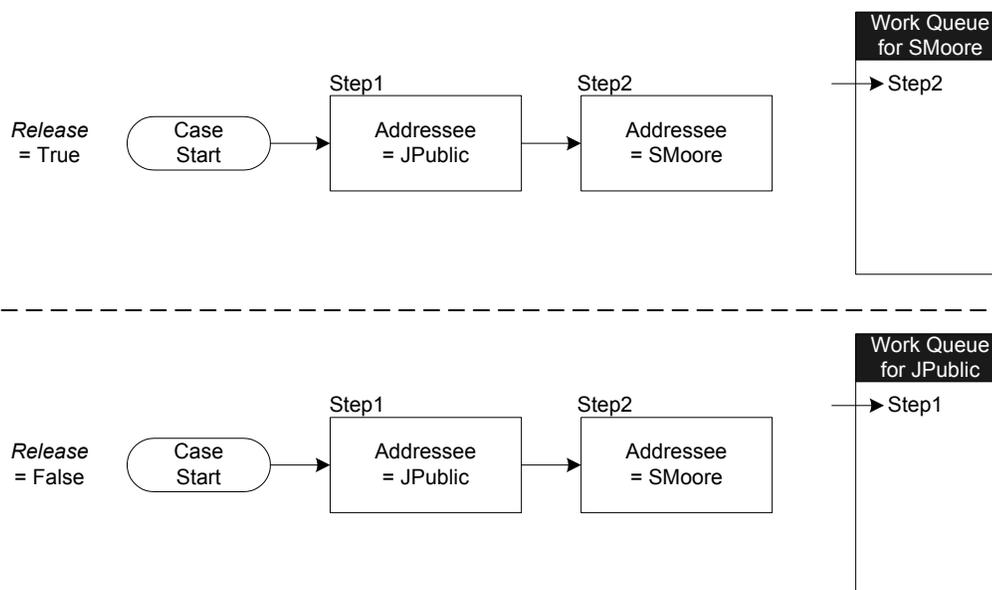
The addressee must be defined in one of the following ways on the Step Definition Addressee Tab in the TIBCO iProcess Modeler:

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



The *Release* flag is disabled if you specify a start step (with the *StartStepName* parameter) other than the first step in the procedure.

Starting a Case with Field Data

Starting a case “with field data” involves passing data with the StartCaseEx method that will be used in the start step of the live case. This is done by passing in field names and corresponding values in the *FieldNames* and *FieldValues* parameters, respectively.

An example is shown below.

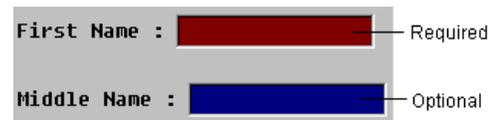
```
Dim sNames(2)
Dim sValues(2)

sNames(0) = "CUSTOMER_NAME"
sNames(1) = "SALARY"
sNames(2) = "MORTGAGE"
sValues(0) = txtCustomer.Text
sValues(1) = txtSalary.Text
sValues(2) = txtMortgage.Text

oProc.StartCaseEx(txtCaseDesc.Text, oProc.StartStepName, False, False, _
    sNames(), sValues())
```

Validating Markings on the Start Step

When a Staffware form is created, the markings that are included on the form are given a “type” designation, indicating whether data in that field 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.



Note - The term “Staffware form” is a remnant of the original software created by Staffware, which was purchased by TIBCO. This term is still used in the TIBCO iProcess Engine and TIBCO iProcess Modeler dialogs and documentation, and will be used here until the engine and modeler dialogs are changed.

This type designation that is assigned to the field when it is added to the form in the TIBCO iProcess Modeler is stored in the **SWFMarking.Type** property for that particular marking on the form. The available marking types are shown below.

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

The StartCaseEx method contains a *Validate* parameter that allows you to specify whether or not to validate the markings on the Staffware form in the start step, based on the marking types defined on the Staffware form:

- If *Validate* = **True**, 1) Validate that the markings exist on the form, 2) validate that all required markings (swRequired) on the form are sent to the server with non-empty data, and 3) validate that display markings (swDisplay) are not sent to the server. Markings are sent to the server upon case start using the *FieldNames/FieldValues* parameters.
- If *Validate* = **False** (the default), it bypasses the enforcement of marking types on the Staffware form.

This is probably most relevant when you are starting a case with field data and the *Release* flag is set to True (see the previous sections).

Sub-Procedure Precedence

The **StartCaseEx** method provides a *SubProcPrecedence* parameter that allows you to specify the order in which sub-procedure versions will be 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 versions be looked for in a specific order:

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

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

The default is to only look for released versions of sub-procedures. Therefore, if the StartCase method does not include the *SubProcPrecedence* parameter is used, only a released version of the sub-procedure will be started.

If the specified (or default) versions 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 and rebuilding the list of work items, 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 through TIBCO iProcess Objects, 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 **StartCaseEx** method
- In the **SWCase.CaseNumber** property
- In the **SW_CASENUM** system field

The availability of the case number depends, however, on which TIBCO iProcess 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 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 is determined by how frequently the background process "wakes up" and processes instructions from the TIBCO iProcess Objects Server.)

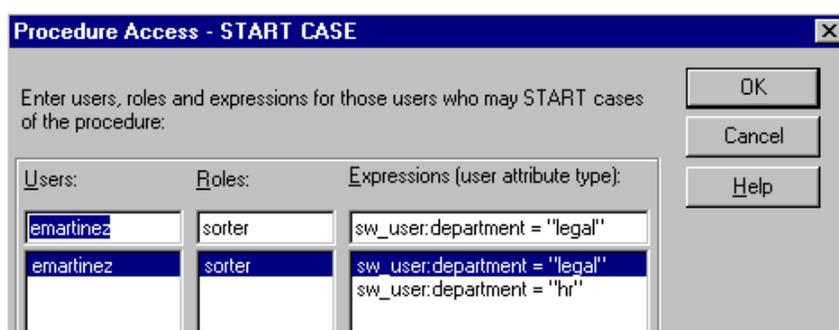
If you are in a situation where you need the case number before the background process can provide it, the following can be used as a work around: A "case number synchronization" step could be added to the procedure definition just after the procedure start step. The addressee for the "case number synchronization" 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 (SWWorkItem.Case.CaseNumber) 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 to the customer's unique number.

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 also specify the users who can start a case of a procedure by using the **Procedure | Access | Case Start** function in the TIBCO iProcess Modeler:



If a user name or role is specified using this function in the TIBCO iProcess Modeler, case-start access to this procedure is limited to only the users, groups, or roles specified, or 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” (SWProc.Status = “swUnreleased”), it can be started only by the procedure owner and user **swadmin**. 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 **swadmin**, can start cases of that procedure.*

You can determine which users, groups, or roles have permission to start a procedure by looking at the **StartByUserRef** property on **SWProc**. (This cannot be set using TIBCO iProcess Objects, however.)

The **StartByUserRef** property returns an **SWAccessUserRef** object, which contains the following properties:

- **UserNames** - A list of the users or groups who have authority to start a case of the procedure.
- **RoleNames** - A list of the roles that have authority to start a case of the procedure.
- **Expressions** - A list 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 properties are empty, *all* users have authority to start a case of the procedure.

Which Procedures can a User Start?

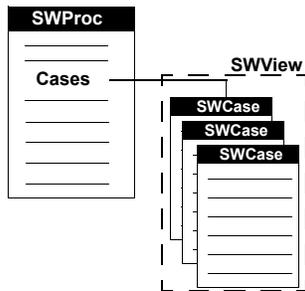
You can also determine the procedures for which a user can start a case by using the **StartProcs** property on the **SWUser** object:

```
'Get the procedures logged-in-user can start cases of
'and display in list box lboStartProcs
'
Set oUser = oNode.LoggedInUser
lblUser.Caption = oUser.Name

lboStartProcs.Clear
For Each oProc In oUser.StartProcs
    lboStartProcs.AddItem (oProc.Name)
Next
Exit Sub
```

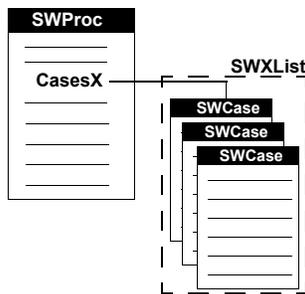
Obtaining the “Default” View / XList of Cases

As cases of a procedure are started and finished, a list of these live cases is maintained by the TIBCO iProcess Engine. When you get an SWProc object from the node, it's a snapshot of the cases for that procedure at that instant. The SWProc object contains two properties that return a list of the cases for that procedure: one as an SWView, the other as an SWXList:



SWProc.Cases - The **Cases** property on SWProc contains a view (SWView) of all of the cases of the procedure that match the filter criteria in the SWView.FilterExpression property. They are listed in the view in the order specified by the SWView.SortFields property.

*Note - Lists of cases obtained with the **Cases** or **CasesX** properties 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.*



SWProc.CasesX - The **CasesX** property on SWProc contains an XList (SWXList) of cases in blocks of size SWXList.ItemsPerBlock of the procedure that match the filter criteria specified in the SWCriteriaC.FilterExpression property. They are listed in the XList in the order specified by the SWCriteriaC.SortFields property.

The following is an example using the default view of cases on a procedure:

```
Dim i As Integer

Set oProc = oNode.LoggedInUser.AuditProcs.Item(lboProcs.ListIndex)

'Get case list for the selected procedure
Set oCases = oProc.Cases

'Add a sortfield to the cases view
oCases.SortFields.Clear
Set oSortField = New SWSortField
oSortField.FieldName = txtSortField
oSortField.IsAscending = optAsc
oCases.SortFields.Add oSortField

'Ask for audit data for the whole list of cases
oCases.IsWithAuditData = True

'Set the case list's filter criteria
oCases.FilterExpression = sFilterCriteria

'Rebuild list of cases
```

```

oCases.Rebuild

'Iterate through the list of cases
While Not (oCases.IsEOL)
    Set oCase = oCases.Item(i)
    i = i + 1
Wend

```

Status of Cases

The **Status** property on SWView and SWXList *always* returns **swChanged** and **swXLChanged**, respectively, for views/XLists of cases. This is true whether the view/XList of cases was obtained with **Cases**, **CasesX**, **MakeViewCases**, or **MakeXListCases**. (See the next section for information about creating a view/XList of cases with the MakeViewCases and MakeXListCases methods.)

Creating an “Alternate” View / XList of Cases

As described in the previous section, the “default” SWView and SWXList of cases is available in Cases and CasesX, respectively, on SWProc — this is the view/XList you will typically use. However, if you have a need to view a list of cases filtered or sorted in more than one way, an additional view/XList of the cases can be created. For example, you may want to view a list of all cases that were started on a particular date (SW_STARTEDDATE=*DesiredDate*) and concurrently view a list of all currently active cases for the procedure (SW_STATUS=“A”). To do this, you would create an additional SWView or SWXList containing SWCase objects filtered and sorted in the desired way.

Alternate SWViews

The following method is available to create alternate views of cases:

- **MakeViewCases** - Creates an additional SWView containing SWCase objects.

This method is available on several objects. The object from which you invoke the method governs the scope of the method. The table below shows the scope of the MakeViewCases method from the objects from which it can be called.

Method	Called From	Scope
MakeViewCases	SWProc	Used to view cases in the procedure.
	SWUser	Used to view cases across multiple procedures on the same node.
	SWEntUser	Used to view cases across multiple procedures on more than one node.

Implied Sort on Alternate Views

If you create an alternate view of cases that spans multiple procedures, there is an implied primary sort. Cases are always sorted by procedure name before any user-defined sort criteria are invoked.

Alternate SWXLists

The following method is available to create alternate XLists of cases:

- **MakeXListCases** - Creates an additional SWXList containing SWCase objects.

This method is only available on SWProc, i.e., it cannot scan multiple procedures or nodes (like `MakeViewCases` — see the previous section).

Status of Cases

The **Status** property on SWView and SWXList *always* returns **swChanged** and **swXLChanged**, respectively, for views/XLists of cases. This is true whether the view/XList of cases was obtained with **Cases**, **CasesX**, **MakeViewCases**, or **MakeXListCases**.

Determining the Number of Cases in a Procedure

There are a number of counts available that tell you how many cases are in a procedure. These counts are available from different objects:

- **SWView** - When the cases are in an SWView, the following counts are available:
 - **Count** - The number of items that are currently stored in the SWView at the client.
 - **ExcludeCnt** - Contains the number of cases that were not included in the indexed collection at the server because they did not satisfy the criteria in the FilterExpression property. (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)
 - **InvalidCnt** - Contains the number of cases not included in the indexed collection because they were invalid within the context of the filter criteria (e.g., the filter expression references a field name not defined in all cases). (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)

Note - The Count, ExcludeCnt, and InvalidCnt properties on SWView are meaningful only after all applicable cases have been evaluated (i.e., IsEOL=True). See “Determining the Number of Objects in a List or View” on page 58 for more information.

- **SWXList** - When the cases are in an SWXList, the following counts are available:
 - **ItemCount** - The total number of items at the server. This count includes only those items that satisfy the filter criteria.

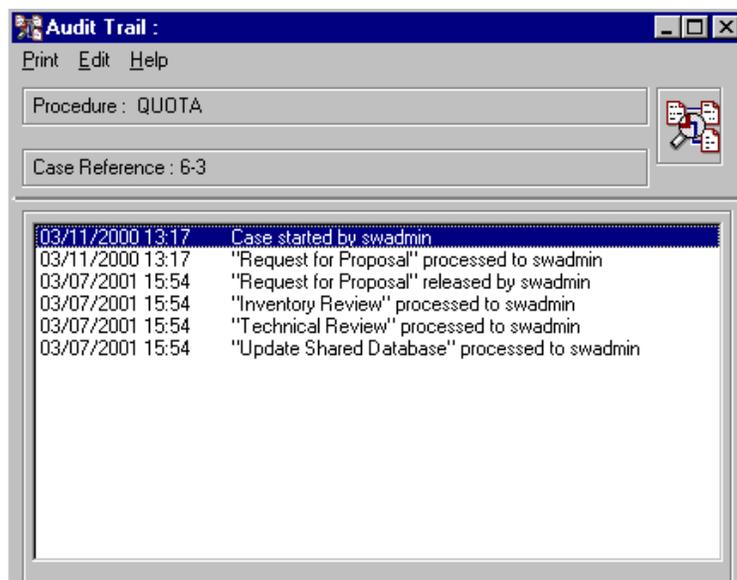
This count is available immediately upon creating the XList (unlike on a view where you need to loop through the list until IsEOL is true to determine the total number of items available on the server).

- **Count** - The number of items that are currently stored in the XList at the client.
- **ExcludeCnt** - Contains the number of cases that were not included in the indexed collection at the server because they did not satisfy the criteria in the FilterExpression property. (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)

- **InvalidCnt** - Contains the number of cases not included in the indexed collection because they were invalid within the context of the filter criteria (e.g., the filter expression references a field name not defined in all cases). (Note - This count may or may not be available, depending on which filtering enhancements have been incorporated in your TIBCO iProcess Objects Server. See the appropriate *Filtering Work Items and Cases* chapter on [page 98](#), [page 126](#), or [page 152](#).)
- **SWProc** - Accessing the following counts on SWProc cause a message to be sent to the server to obtain the count from the procedure:
 - **CaseCnt** - The total number of cases in the procedure.
 - **ActiveCnt** - The number of active cases (SWCase.IsActive=True) in the procedure.
 - **ClosedCnt** - The number of closed cases (SWCase.IsActive=False) in the procedure.

Auditing Case Data

The system maintains an “audit trail” that provides information about the progress through a case, that is, which steps in the case 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 Client):



There are two types of audit trail entries:

- **System-defined** - These are added to the audit trail by the system. These messages are pre-defined in *\$WDIR/etc/language.lng\audit.mes* (Windows) or *\$SWDIR/etc/language.lng/audit.mes* (UNIX). An excerpt from the audit.mes file is shown below:

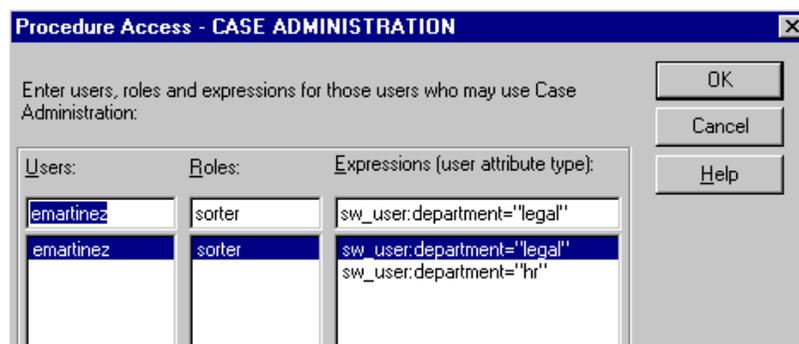
```
000:Case started by %USER
001:"%DESC" processed to %USER
002:"%DESC" released by %USER
003:Deadline for "%DESC" expired for %USER
:
:
```

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 **AddAuditEntry** method. 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 246](#).

Determining the Procedures a User can Audit

Before a user can audit cases, that user must have the proper authority. Within the procedure definition, you can specify the users who can audit the procedure by using the **Procedure | Access | Case Admin** function in the TIBCO iProcess Modeler:



This dialog is used to define who has **case administration** authority. If a user, group, or role name is specified using this dialog, 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 **swadmin** user always has access to the audit trail of any procedure.) If no one is designated as having access through this function, it defaults to giving *everyone* access.

You can determine which procedures a user can audit by using the **AuditProcs** property on **SWUser**. This property returns an SWList of **SWProc** objects, one for each procedure the user has permission to audit. See the example below.

```
'Display in the lboProcs list box the procedures on this node
'that the logged-in-user can audit
'
With lboProcs
  .Clear
  For Each oProc In oNode.LoggedInUser.AuditProcs
    lboProcs.AddItem (oProc.Name)
  Next
End With
```

Populating a Case with Audit Data

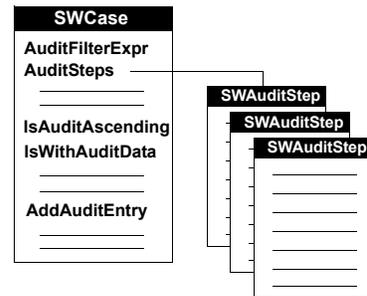
Information about which steps in a case have been processed (audit data) is maintained on the TIBCO iProcess Engine. This information is not acquired by default — you must explicitly ask for it by setting the **IsWithAuditData** property to true. This property can be found on the following objects:

- **SWCase** - Setting **IsWithAuditData** to true on this object causes audit data to be maintained in the **individual case**.
- **SWView** - Setting **IsWithAuditData** to true on this object causes audit data to be maintained in **all cases in the view**.
- **SWCriteriaC** - Setting **IsWithAuditData** to true on this object causes audit data to be maintained in **all cases in the XList**.

```
'Return audit data on all cases in the view
Set oProc = oNode.LoggedInUser.AuditProcs.Item(lboProcs.ListIndex)
Set oCases = oProc.Cases
oCases.IsWithAuditData = True
oCases.Rebuild
```

Once you have specified that audit data is to be maintained in a case (or all of the cases in a view or XList), then rebuilt the case, view, or XList, the **AuditSteps** property on **SWCase** will contain an **SWList** of **SWAuditStep** objects, one for each step that has been processed in the case. (Note that if you are rebuilding an **SWCase** object to populate **AuditSteps**, you must also set the **IsRebuildAll** flag to True to also get the subordinate objects.)

Note - It's possible the list of audit steps on the case is being filtered, in which case, it may not contain ALL steps that have been processed. It may also contain custom entries. Audit data filtering and custom audit entries are described later in this chapter.

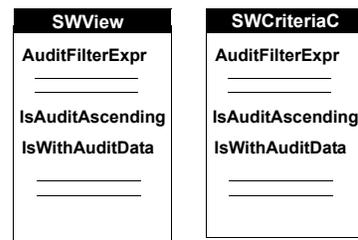


The **IsAuditAscending** property allows you to specify the chronological order in which the audit data is returned from the server — a value of True (default) causes audit data to be returned in chronologically ascending order. False causes audit data to be returned in chronologically descending order.

The **SWView** and **SWCriteriaC** (for XLists) objects also contain properties that allow you to specify that all cases on the view or XList are to maintain audit data:

Note that it is preferable to get audit data only on the cases for which it is needed. This functionality is resource expensive:

- If you have a list of cases in a view or XList and you need audit data on all or most of the cases, it is much more efficient to request audit data for the entire view or XList by setting the **IsWithAuditData** property on the **SWView** or **SWCriteriaC** object.
- If you need audit data for only one or a few of the cases in a view or XList, it is much more efficient to request audit data by setting the **IsWithAuditData** property on the individual **SWCase** object.



The SWAuditStep Object

Each **SWAuditStep** object in the **AuditSteps** property list represents a single action that has been processed in the case. The **Action** property identifies what this action is, based on the **SWAuditActionType** enumeration.

The **Message** property contains 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 239](#) for information about these files). For example, if the message in **audit.mes** is:

```
%DESC processed to %USER
```

the string in the **Message** property 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** property on **SWAuditStep** can be used to determine if the step has been “processed to” the addressee of the step that the **SWAuditStep** object represents, but not yet released to the next step in the procedure. In other words, it is the step at which the case is currently sitting.

SWAuditStep	SWAuditActionType
getAction	swStartCase = 0
getClassId	swProcessedTo = 1
getDate	swReleasedBy = 2
getDescription	swDeadlineExp = 3
getKey	swForwarded = 4
getMessage	swProcessedFor = 5
getName	swError = 6
getProcMajorVer	swTermAbnormal = 7
getProcMinorVer	swTermPremature = 8
getSubCaseId	swTermNORMAL = 9
getTimeOffset	swRevisedBy = 10
getUser	swReleasedMBox = 11
isOutstanding	swModifiedBy = 12
	swDeadlineWdl = 13
	swResent = 14
	swEventIssued = 15
	swSubCaseStart = 16
	swSubCaseComp = 17
	swSubCaseTerm = 18
	swSubCaseExpired = 19
	swSubCaseWithdrawn = 20
	swRedirectedTo = 21
	swSuspendedBy = 22
	swResumedBy = 23
	swCaseJumpBy = 24
	swDynaGraftCaseStart = 25
	swTaskCountSet = 26
	swTaskDeleted = 27
	swSubCaseGrafted = 28
	swExtProcessGrafted = 29
	swGraftInitiated = 30
	swExtProcessReleased = 31
	swGraftReleased = 32
	swDynamicReleased = 33
	swCaseMigrated = 34
	swGraftWithdrawn = 35
	swDynaGraftDeadlineExp = 36
	swDynamicWithdrawn = 37
	swKeepOnWithdraw = 38
	swReleasedNoAddressees = 39
	swReleasedNoSubProcs = 40
	swEAIcallInitiated = 50
	swEAIcallComplete = 51
	swEAIcallExpired = 52
	swEAIcallWithdrawn = 53
	swTransProcessed = 54
	swTransStarted = 55
	swTransRestart = 56
	swWIOpenedBy = 59
	swWIKeptBy = 60
	swEAIcallFailed = 80
	swErrMaxActions = 81
	swErrGenericTransfail = 82
	swEAINoPlugin = 83
	swErrBadSubProc = 84
	swErrDiffTemplate = 85
	swErrDiffTemplateVer = 86
	swTransAborted = 87

Configuring Audit Trail Strings

As you can see from the example above, 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 **SWAuditStep** 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 SWAuditStep 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"
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 also a number of audit action types that are specific to sub-procedures. These are enumerated in **SWAuditActionType**:

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.

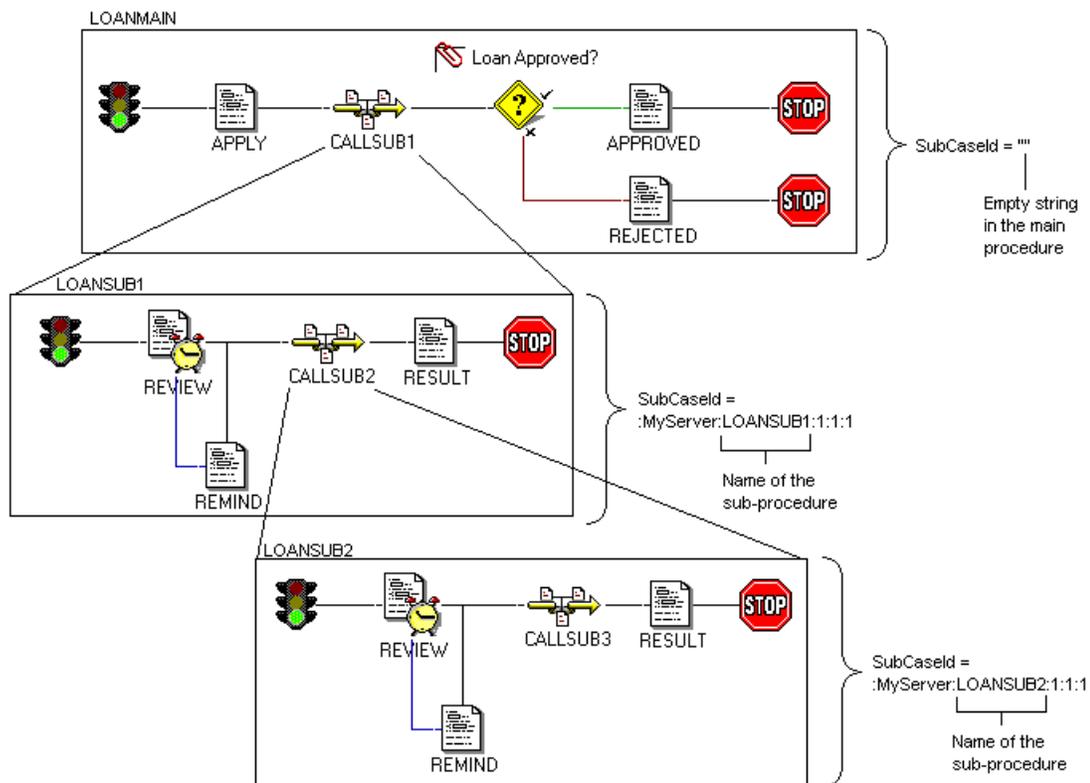
The **SWAuditStep** object also contains a **SubCaseId** property that returns a system-generated string that identifies whether or not the audit step or sub-procedure call step is in the main procedure or a sub-procedure. This property may be used to group audit steps by case when steps from both main procedures and sub-procedures are being processed simultaneously.

If the audit step/sub-procedure call step is in the main procedure, the SubCaseId contains an empty string. If it is in a sub-procedure, the SubCaseId contains the name of the sub-procedure.

Note that additional information is also returned from the TIBCO iProcess Engine in the SubCaseId. This requires that you parse this string to obtain the name of the sub-procedure. An example is shown below:

```
SWAuditStep.SubCaseId = :MyServer:LOANSUB1:1:1:1
```

The LOANSUB1 part of the SubCaseId identifies the name of the sub-procedure, i.e., this particular audit step is from sub-procedure LOANSUB1. Also see the illustration below.



For more information about auditing, see [“Auditing Case Data” on page 239](#).

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 setting a filter criteria expression in the **AuditFilterExpr** property. If the **IsWithAuditData** property is set to True, only SWAuditStep objects that satisfy this filter expression are returned from the server and placed in the AuditSteps list.

The AuditFilterExpr property can be found on the following objects:

- **SWCase** - This allows you to set AuditStep filter criteria on an individual case.
- **SWView** - This is used to set AuditStep filter criteria for all cases in the view.
- **SWCriteriaC** - This is used to set AuditStep filter criteria for all cases in the XList.

If you need to filter audit steps on one or just a few cases, it may be more efficient to set the AuditFilterExpr property on the individual case(s), then rebuild the SWCase.AuditSteps list on those cases. Whereas, if you need to filter audit steps on most or all of the cases in a view or XList, it is more efficient to set the AuditFilterExpr property on the SWView or SWCriteriaC object, then rebuild the SWView or SWXList.

The initial value of AuditFilterExpr on a SWCase object defaults to the AuditFilterExpr property on the view or XList that holds the case. The value is taken at the time the case is created, so changing the property on the SWView or SWCriteriaC of an XList at a later time will not change the property values on the individual cases until the view or XList is rebuilt.

Setting AuditFilterExpr

To set audit data filter criteria through the AuditFilterExpr property, 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.

An example is shown below:

```
oSWCase.AuditFilterExpr = "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 [].

Syntax details for each of the filter criteria are provided in the *AuditFilterExpr* property topic in the on-line help system.

Adding User-defined Audit Trail Entries

You can add user-defined audit trail entries to a live case by invoking the `SWCase.AddAuditEntry` method.

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 `AddAuditEntry` method.

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

eventnum:event_description

where:

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

event_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 `AddAuditEntry` method (see the example below).

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 `AddAuditEntry` 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.)

An example is shown below of adding a user-defined audit entry of *eventnum* (MsgID) 256 to the audit trail of a step:

```
oWorkItem.Case.AddAuditEntry oWorkItem.StepName, _
oWorkItem.StepDescription. oUser.Name, 256
```

For syntax details for `AddAuditEntry`, see the on-line help system.

Be aware that entries you add via the `AddAuditEntry` 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.

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" (**SWPredictedItem** 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.

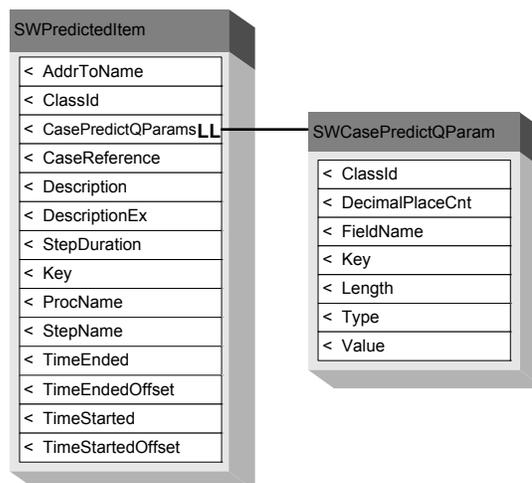
Included with the work items returned is information about the expected times the work items are predicted to start and end, providing information that can be used to predict the outcome of the case. This can be used to improve work forecasting and estimate the expected completion of cases.

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 **MakeXListPredict** method. See [“Background Case Prediction” on page 249](#) 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 249](#) for more information.
 - *Case simulation*, 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 250](#) for more information.



Defining Case Prediction

The following subsections describe how case prediction is defined for a procedure using the TIBCO iProcess Modeler.

Step Duration

As you are defining a procedure using the TIBCO iProcess Modeler, 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 the TIBCO iProcess Modeler. A prediction duration can be defined for the following types of steps:

- Normal step
- Event set
- EAI step
- Sub-procedure call step
- Dynamic sub-procedure call step
- Graft step

The step's duration definition is represented by an **SWDuration** object, which is returned by the **SWStep.Duration** property.

The **SWDuration** object contains the following properties:

- **DurationValues** - This property returns a list of **SWDurationValue** objects, which provide access to the values that were entered in the Deadline/Step Duration Definition dialog for that step.
- **Type** - This property indicates (using the **SWDurationType** enumeration constant) 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.

Note that if the step deadline is used for the duration (**SWDuration.Type = swDurationDeadline**), the duration value will NOT be found in the **SWDurationValue.Value** property. Rather, it will be found in **SWDeadlineValue.Value**.

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 represented by the **SWStep.IsPrediction** property.

The **IsPrediction** property 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 calling step. If a duration is defined for the sub-procedure, it is accessible with the **SWProc.Duration** property.

Conditional Actions for Case Predictions

The "conditional" definition for a step in the TIBCO iProcess Modeler 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 the TIBCO iProcess Modeler is represented by the **SWConditional.PredictType** property. This property returns an enumeration constant (**SWConditionPredictType**) that identifies how the "prediction condition" was defined in the TIBCO iProcess Modeler.

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 **SWNode.MakeXListPredict**. This method returns an **SWXList** of **SWPredictedItem** 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 the TIBCO iProcess Modeler, the **Prediction** flag on the **Procedure Status** dialog must be checked to enable prediction on this procedure. This flag can be accessed in the **SWProc.IsPrediction** property. This property returns True if prediction has been enabled for the procedure.

The **SWPredictedItem** objects returned in the **SWXList** can be filtered and/or sorted using the **FilterExpression** and **SortFields** properties on the **SWCriteriaP** object (see [“Filtering and Sorting Predicted Items” on page 252](#)). You can also filter or sort on case data in CDQP fields if they have been configured properly for prediction (see [“Including Case Data Queue Parameter Data in Prediction Results” on page 251](#) for more information).

You can also persist the **SWXList** returned by **MakeXListPredict** by setting the **SWCriteriaP.IsPersisted** property to True. You must then save the persistence ID (**SWCriteriaP.PersistenceID**) and use it at a later time as an input parameter with the **GetXListPredict** method to retrieve the same **SWXList** of **SWPredictedItem** objects. (For more information about persisting an **XList** of predicted work items, see [“Working with Persisted XLists” on page 81](#).)

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 **SWProc.PredictCase**. This method returns an **SWList** of **SWPredictedItem** 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 the TIBCO iProcess Modeler (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 **SWProc.SimulateCase**.

This method returns an **SWList** of **SWPredictedItem** 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 will start 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 the TIBCO iProcess Modeler (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 MakeXListPredict**
 - **SWPredictedItem** objects will be returned for sub-procedure call steps that have not yet been processed (the process flow has not reached the step).
 - **SWPredictedItem** objects will NOT be returned for sub-procedure call steps that are currently outstanding (the process flow has reach the step and its sub-procedure has been started). However, an **SWPredictedItem** object will be returned for each outstanding item in the sub-procedure that has been started by the sub-procedure call step.
- **SimulateCase**
 - **SWPredictedItem** objects will NOT be returned for sub-procedure call steps. However, an **SWPredictedItem** 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 MakeXListPredict**
 - **SWPredictedItem** 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).
 - **SWPredictedItem** 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 **SWPredictedItem** 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 pre-

diction 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**

- SWPredictedItem 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).

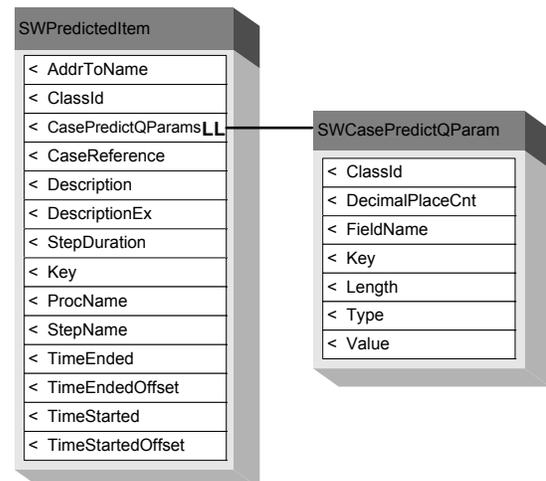
Including Case Data Queue Parameter Data in Prediction Results

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 represented by the **SWCaseDataQParamDef.IsPrediction** property 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 **SWCasePredictQParam** object representing the CDQP field is returned on the SWPredictedItem object. All CDQP fields returned are available from SWPredictedItem in the **CasePredictQParams** property. These SWCasePredictQParam objects provide access to the case data in the predicted work items.

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 as SWCasePredictQParam objects.
- SWCasePredictQParam objects are returned only on SWPredictedItem objects that represent normal steps.
- SWCasePredictQParam 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 MakeXListPredict is called.)



The SWCasePredictQParam objects returned in this property 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 MakeXListPredict method (which returns an SWXList; filtering and sorting can't be performed on SWList objects, which are returned by the PredictCase and SimulateCase methods).

Filtering and Sorting Predicted Items

You can filter and/or sort the results of the **makeXListPredict** method, which are returned in an SWXList. This is done by using the **FilterExpression/getFilterExpression** and **SortFields/getSortFields** on **SWCriteriaP**. 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 152](#) for more information. Also see the *Sorting Work Items and Cases* chapter on [page 178](#) for information about how to sort items that are returned in a pageable list.

The *Filtering Work Items and Cases* chapters and the *Sorting Work Items and Cases* chapter provide lists of system fields that can be used when filtering / sorting. Note, however, that when filtering and sorting **SWPredictedItem** 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
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") ^a
SW_ARRIVALDATE_USECS	X		Numeric
SW_LEAVE_DATE	X		String ("YYYY-MM-DD HH:MM:SS") ^a
SW_LEAVE_DATE_USECS	X		Numeric

a. The hour component is in 24-hour format.

You can also filter on CDQP fields that are in the work items returned by MakeXListPredict, 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 the filter expression, an error is not returned. Instead, the filter operation will not return any items.

Triggering Events

An Event step is a special kind of step that is processed by calling the **TriggerEvent** method on SWCase. 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 *FieldName / FieldValue* pairs to 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 91](#) for information about the difference between case data and work item data.

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

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

Suspending Cases

You can suspend activity in a case family (a main case and all of its sub-cases) using the **SetState** method on **SWCase**. The **SetState** method allows you to set the state of a case family to either **suspended** or **active** (used to “unsuspend” a case family).

To suspend a case, call the **SetState** method, passing **swSuspended** (an **SWCaseStateType** enumeration) in the *NewState* parameter.

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 with this method.
- 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 with this method. 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.

When the state of a case family is set with the **SetState** method, the **IsSuspended** property on **SWCase** is automatically set; it is set to True if the case family is suspended, or False if the case family is active. This property is also available on the **SWWorkItem** object, representing the work items in the case family.

When calling the **SetState** method, you can optionally pass *FieldName / FieldValue* pairs to 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 91](#) for information about the difference between these types of data.

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 **SetState** method, passing **swActive** (an **SWCaseStateType** enumeration) in the *NewState* 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 Suspended Cases

When a procedure is created with the TIBCO iProcess Modeler, 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 **IsIgnoreState** property on **SWStep**.

Jumping To New Outstanding Step in a Case

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 the new 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 **SWCase** object. The **JumpTo** method allows you to:

- Specify the set of outstanding items/steps 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 specify a ‘*’ wildcard character 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 91](#) for information about the difference between these types of data.)

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

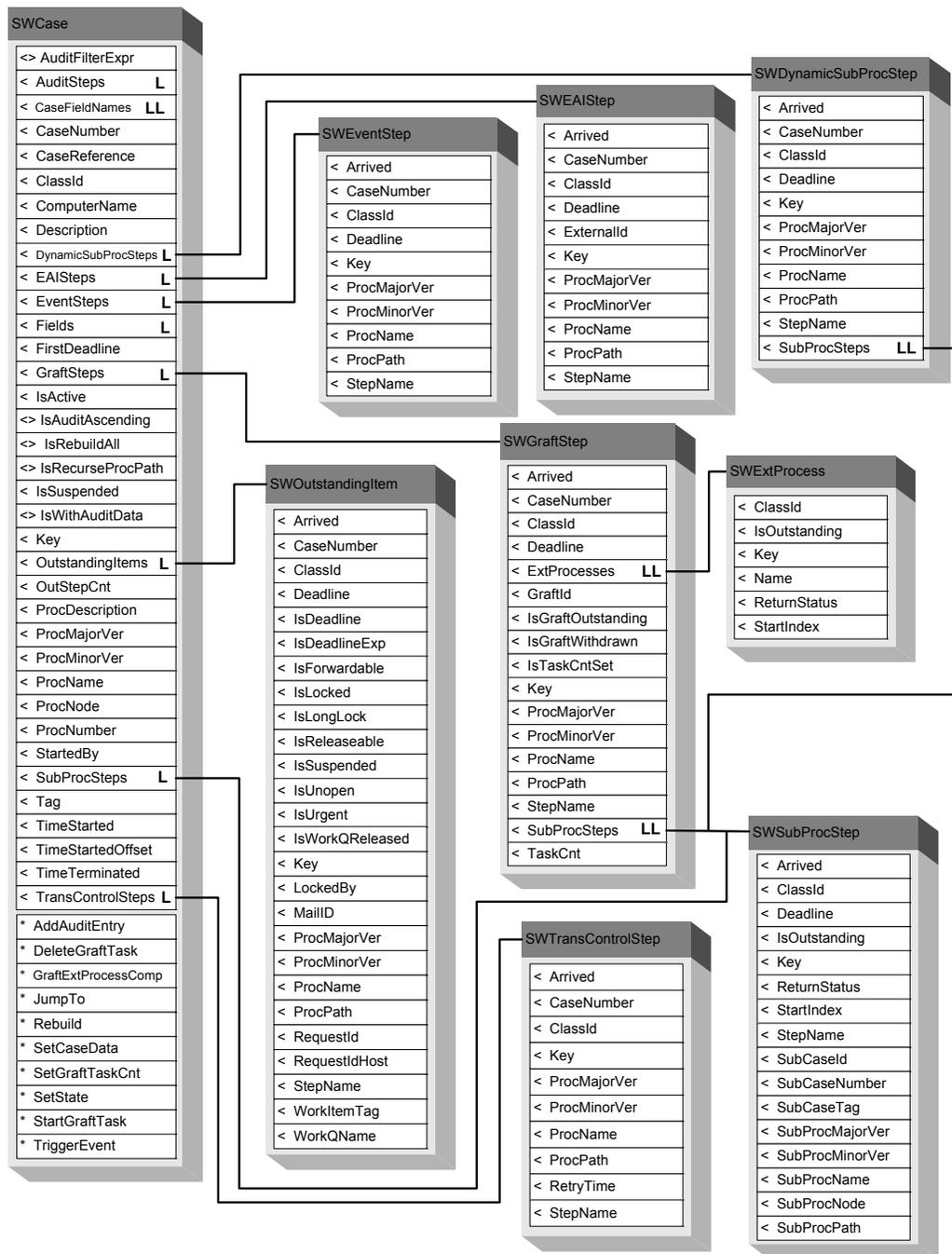
The **SWCase** object provides properties that return lists of each of the different types of steps that are currently outstanding in the case family. Obtaining a list of the current outstanding items in a case family allows you to determine the items that are available for withdrawal with the **JumpTo** method.

The properties on **SWCase** that return outstanding item objects are:

- **OutstandingItems** - This returns a list of **SWOutstandingItem** objects, one for each normal step that is outstanding in the case family.
- **EventSteps** - This returns a list of **SWEventStep** objects, one for each event step that is outstanding in the case family.
- **EAISteps** - This returns a list of **SWEAIStep** objects, one for each EAI step that is currently outstanding in the case family.
- **SubProcSteps** - This returns a list of **SWSubProcStep** objects, one for each sub-procedure call step that is currently outstanding in the case family. Note that this property is also available on the **SWDynamicSubProcStep** and **SWGraftStep** objects; from these objects, the **SubProcSteps** property returns **SWSubProcStep** objects that represent the *actual sub-procedures* that were started from the dynamic sub-procedure call step or graft step. Your withdraw list can include either a sub-procedure call step that is outstanding in a case, or a sub-procedure that was started by a dynamic sub-procedure call step or graft step. If you specify the name of a sub-procedure that was started by a dynamic sub-procedure call step or graft step, all outstanding items in that sub-procedure are withdrawn.
- **DynamicSubProcSteps** - This returns a list of **SWDynamicSubProcStep** objects, one for each dynamic sub-procedure call step that is currently outstanding in the case family.
- **GraftSteps** - This returns a list of **SWGraftStep** objects, one for each graft step that is currently outstanding in the case family. (Note that graft steps can result in outstanding items, although you cannot withdraw this type of outstanding item.)
- **TransControlSteps** - This returns a list of **SWTransControlStep** objects, one for each outstanding transaction control step in the case. (Note that transaction control steps can result in outstanding items, although you cannot withdraw this type of outstanding item.)

The **IsRecurseProcPath** property on **SWCase** specifies whether or not outstanding items in sub-cases launched from the **SWCase** are also returned in the properties listed above. Setting this property to **True** causes outstanding steps from the main case as well as all sub-cases to be returned. Setting it to **False** (the default) causes the outstanding steps to be returned only from the main case.

The following illustrates the outstanding item objects available in the object model.



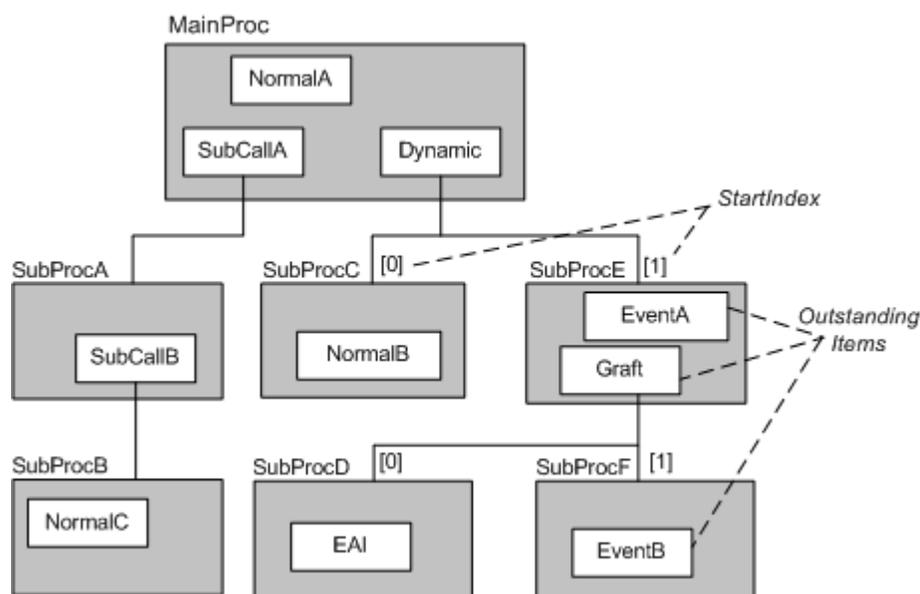
ProcPath to Outstanding Items

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. Note that the ProcPath may point to either an outstanding item/step, or to an outstanding sub-procedure, in the case of sub-procedures that are started by dynamic sub-procedure call steps and graft steps. (If you withdraw a sub-procedure that was started by a dynamic sub-procedure call step or graft step, all outstanding items in that sub-procedure are withdrawn.)

The ProcPath for an outstanding item can be obtained by accessing the following properties:

- **ProcPath** - This property is on the SWOutstandingItem, SWEventStep, SWEAISTep, SWGraftStep, and SWDynamicSubProcStep objects. It returns a string that provides the path from the main procedure to the outstanding item.
- **SubProcPath** - This property is on the SWSubProcStep object. It provides the path from the main procedure to the outstanding sub-procedure call step or the sub-procedure (if the SWSubProcStep object represents a sub-procedure started by a dynamic sub-procedure call step or graft step).

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	Sub-procedure	ProcPath
NormalA	"NormalA"	SubProcC	"Dynamic[0]"
NormalB	"Dynamic[0] NormalB"	SubProcD	"Dynamic[1] Graft[0]"
NormalC	"SubCallA SubCallB NormalC"	SubProcE	"Dynamic[1]"
EventA	"Dynamic[1] EventA"	SubProcF	"Dynamic[1] Graft[1]"
EventB	"Dynamic[1] Graft[1] EventB"	Note that since SubProcA and SubProcB are started by sub-procedure call steps, they do not result in SWSubProcStep objects. Therefore, there is no ProcPath to those sub-procedures. Those sub-procedures can be withdrawn by withdrawing their sub-procedure call steps (SubCallA and SubCallB).	
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).

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 any dynamic sub-procedure call steps or graft steps that start multiple sub-procedures (see the Dynamic or Graft step in the example above), 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. In addition to appearing in the ProcPath 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 accessing the **StartIndex** property on the **SWSubProcStep** object.

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

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 that 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*
- an external 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 (**TaskCnt**) has reached zero (the task count is decremented when a task is 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 the TIBCO iProcess Modeler. A graft step is represented by the **SWGraftStep** object. The following are elements of a graft step definition:

- **Sub-Procedures / External Processes to Start** - When a graft step is defined in the TIBCO iProcess Modeler, 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", which can contain multiple elements that are accessed

by index. At run-time, the external application will write the names of the sub-procedures and external processes to start into the elements of this array field.

The name of the array field containing the names of the sub-procedures and/or external processes to start is available in the **SWStep.SubProcName** property.

See [“Array Fields” on page 94](#) for information about how array fields are used with graft steps.

- **Return Status** - When a graft step is defined in the TIBCO iProcess Modeler, you can specify a numeric array field, whose elements will contain a return status for each corresponding sub-procedure in the SubProcName array field. The name of the array field containing the return statuses is reflected in the **SWStep.SubProcStatus** property. See [“Return Statuses” on page 262](#) for more information about return statuses.

Starting a Graft Task

A graft task is started with the **StartGraftTask** method on **SWCase**. In this method call, you must list the names of the sub-procedures and/or external processes that are being started as part of this task initiation. These names 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 TIBCO iProcess Engine will keep track of which of those sub-procedures have completed.
- If **external processes** are started with the StartGraftTask method, the client application must keep track of when those external processes have completed, and for each one, must inform the engine by calling the **GraftExtProcessComp** method when it is complete. (Note that calling the StartGraftTask method does not actually start these external processes. It is merely providing the client the names of external processes that must be completed before the graft task is considered complete.)

After specifying the names of the external processes to start in the StartGraftTask method, the names are available in the **SWGraftStep.ExtProcessNames** property. External processes are no longer outstanding and are removed from this list when you call GraftExtProcessComp to tell the engine that the external process is complete.

Calling the StartGraftTask method causes the graft step's task count (**TaskCnt** -- see the next subsection) to be decremented by one.

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 for 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.)

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 that can be passed to sub-procedures that are started by the StartGraftTask method.

Setting the Task Count

The client application must inform the TIBCO iProcess 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** property to be set to True, and increments the graft step's **TaskCnt** property by the number specified in the SetGraftTaskCnt method. (Note that the task count (TaskCnt) may be negative if you start and complete one or more tasks before you call the SetGraftTaskCnt method.)

The task count is automatically decremented by one each time the **StartGraftTask** or **DeleteGraft-Task** method is called.

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, an **SWGraftStep** object representing the outstanding graft item will be returned from the **GraftSteps** property on SWCase.

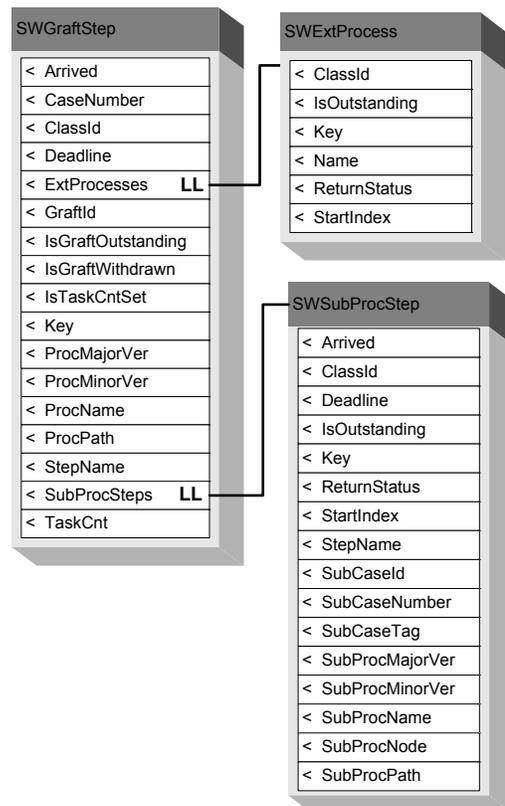
Note that although the graft item is considered “outstanding” (because it is returned when you access GraftSteps), the **IsGraftOutstanding** property on SWGraftStep returns True only if the process flow has reached the graft step in the procedure. It does not return True if the graft item is outstanding because it was initiated with StartGraftTask or SetGraftTaskCnt.

The **IsRecurseProcPath** property on SWCase can be used to specify whether or not the GraftSteps property should be recursive, i.e., should the list include outstanding graft items from sub-procedures that have been launched from the main procedure.

You can determine the sub-procedures and/or external processes that were started by the outstanding graft item by accessing the following properties on **SWGraftStep**:

- **SubProcSteps** - Returns a local list of **SWSubProcStep** 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 accessing the **IsOutstanding** property on SWSubProcStep; this property returns True if the sub-procedure is still outstanding (it hasn't completed yet).

Note that SWSubProcStep objects returned by the SubProcSteps property represent only *sub-procedures* that were started by the graft item (the ProcPath returned by the SubProcPath property on SWSubProcStep points to the sub-procedure).



- **ExtProcesses** - Returns a local list of **SWExtProcess** 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 accessing the **IsOutstanding** property on **SWExtProcess**; 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 **GraftExtProcessComp** 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 accessing the **ReturnStatus** property on **SWSubProcStep** (for sub-procedures) or **SWExtProcess** (for external processes). These are described below:

- **Sub-Procedures** - The **ReturnStatus** property on **SWSubProcStep** 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 **ReturnStatus** 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 **GraftExtProcessComp** method is called to tell the engine that the external process is complete, a user-defined status can be specified in the method call. This *ReturnStatus* parameter can be any integer the user chooses. It defaults to 0.

The **ReturnStatus** property on **SWExtProcess** returns the integer that was specified when the **GraftExtProcessComp** method was called.

*Note - You can also get the return status by using the ReturnStatus array field (**SubProcStatus** on **SWStep**) that was specified in the graft step definition. The elements of the Return Status array fields contain the status code for the corresponding sub-procedures or external processes named in the **SubProcName** array field (**SubProcName** on **SWStep**).*

Other Status Information on an Outstanding Graft Item

The **SWGraftStep** objects that are returned in the **GraftSteps** property provide the following types of status and count information about the graft step:

- whether the task count has been set yet on this graft step -- **IsTaskCntSet** property
- the current task count -- **TaskCnt** property
- the graft step's ID -- **GraftId** property (This will be established for the graft step if **StartGraftTask** or **SetGraftTaskCnt** has been called to provide the graft ID.)
- whether the graft step is withdrawn -- **IsGraftWithdrawn** property
- the graft step's deadline, if set -- **Deadline** property
- the case number of the case containing the graft step -- **CaseNumber** property
- the name and path to the graft step -- **ProcPath** property

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 **TaskCnt** property on **SWGraftStep** 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 -- **SWGraftStep.IsGraftOutstanding** = True
- the graft step's task count has been set -- **SWGraftStep.IsTaskCntSet** = True
- all of the sub-procedures and/or external processes started for the graft step have completed
- all tasks have been completed -- **SWGraftStep.TaskCnt** = 0

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 reflected in the following properties on **SWStep**:

- **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; see the *TIBCO iProcess Modeler Advanced Design Guide* for information about parameter templates.)
- **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 graft 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 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 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 the TIBCO iProcess Modeler 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 the TIBCO iProcess Modeler.

For example, in the following procedure fragment, StepA is followed by a series of EAI steps. The release of StepA and the execution and release of EAI1, EAI2, and EAI3 occurs as a single transaction. A failure at any point in the series causes the transaction to rollback prior to the release of StepA.



By adding a transaction control step (TCS) between EAI2 and EAI3 as in the example below, two transactions are created. When the process flow reaches the transaction control step, the current transaction is committed (assuming the successful execution and release of EAI1 and EAI2), then a second transaction is started. A failure in execution/release of EAI3 will only cause a rollback prior to the execution of EAI3.



Note that this is just one example of how transaction control steps can be used. For information about additional ways transaction control steps can be placed and used in procedures, see the *TIBCO iProcess Modeler Integration Techniques Guide* (which is provided in the documentation library on the distribution CD).

Step Type

The **Type** property on the **SWStep** object will return the following constant if the step has been defined as a transaction control step:

Constant	Description	Value
swTransControl	Transaction Control Step.	'T'

This constant is defined in the **SWStepType** enumeration.

Type of Transaction Control Step

When a transaction control step is defined in the TIBCO iProcess Modeler, 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 in the **TransControlType** property on the **SWStep** object. This property 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'
swNA	Not applicable for this step type.	'N'

These constants are defined in the **SWTransControlType** enumeration. The **TransControlType** property will return **swNA** if the step represented by **SWStep** is not a transaction control step.

Outstanding Transaction Control Steps

The **TransControlSteps** property on **SWCase** returns an **SWList** of **SWTransControlStep** objects, one for each outstanding transaction control step in the case family.

See “[Determining Outstanding Items](#)” on page 256 for more information about obtaining and using outstanding items.

SWTransControlStep
< Arrived
< CaseNumber
< ClassId
< Key
< ProcMajorVer
< ProcMinorVer
< ProcName
< ProcPath
< RetryTime
< StepName

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 properties reflect this retry delay/time:

- **SWStep.RetryDelay** - This property 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.
- **SWTransControlStep.RetryTime** - This property 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 **RetryDelay** 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 **RetryTime** property 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 **RetryTime** property does not reflect these additional retries — it will only contain the date/time of the initial retry — the one controlled by the Deadline Manager.

Transaction Control Step 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 in the **SWAuditStep.Action** property.

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.

Closing Cases

Active cases of a procedure can be closed using the following methods:

- **CloseCases** - This method allows you to close one or more cases identified by their case number(s).
- **CloseByCriteria** - 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 view or XList of cases. For information about the allowable filter criteria, see [“Filtering Work Items and Cases” on page 152](#).

This method is available on both the SWProc and SWNode objects. If called from SWNode, you can optionally provide a list of procedures. Only cases of those procedures that match the filter criteria are closed. (Note that you are not allowed to close cases from a slave node.)

To close cases using either of the methods above, the user must have system administrator authority (MENUNAME attribute must be ADMIN — see [“User Attributes” on page 221](#) for information about the MENUNAME attribute).

After closing a case, the **SWCase.IsActive** property will be False to indicate it is no longer active.

You can filter and sort cases on their “active” or “closed” status by using the SW_STATUS system field. If a case is active, SW_STATUS = “A”; if a case is closed, SW_STATUS = “C”.

Resurrecting a Closed Case

A closed case can be “resurrected” (i.e., it’s status changed to “active”) using the **TriggerEvent** method on SWCase. From the SWCase object for the closed case, call TriggerEvent and pass True in the *Resurrect* 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

Purging cases permanently deletes them from the system. You can purge cases using the following methods:

- **PurgeCases** - This method purges one or more closed cases identified by their case number(s).
- **PurgeByCriteria** - 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 [“Filtering Work Items and Cases” on page 152](#).

This method is available on both SWProc and SWNode. If called from SWNode, you can optionally provide a list of procedures. Only cases of those procedures that match the filter criteria are purged. (Note that you are not allowed to purge cases from a slave node.)

- **PurgeAndReset** - This method purges ALL cases of the procedure and resets the case counter (SWProc.CaseCnt) to 0.

To purge cases using any of the methods above, the user must have system administrator authority (MENUNAME attribute must be ADMIN — see [“User Attributes” on page 221](#) for information about the MENUNAME attribute).

15

Stateless Programming

Introduction

When using TIBCO iProcess Objects in a web-based environment, there are some stateless-model concepts that you need to adhere to for your application to be efficient:

- The **Logout** method should NOT be called every time you disconnect from the server — see the [“Logging Out vs. Disconnecting”](#) section below for more information.
- Use the “Make” methods to avoid pulling down large amounts of data from the server to access a single item — see the [“Stateless Objects” on page 271](#) for more information.
- Use XLists, especially persisted XLists of work items, to minimize network traffic — see [“Persisted XLists” on page 274](#) for more information.

Logging Out vs. Disconnecting

Logging into a TIBCO iProcess Objects Server accomplishes two things:

- Connecting to the TIBCO iProcess Objects Server
- Starting a SAL session

Connecting to the TIBCO iProcess Objects Server is very fast. It is the same connection that is made between your browser and a web server every time you access a web page.

Starting a SAL session, on the other hand, is a very time-intensive operation. It involves connecting to the Work Queue Server, caching bits of information, validating the user name and password, among other things.

Calling the **Login** method always causes a connection to the TIBCO iProcess Objects Server to be established. It then looks to see if a SAL session for the user already exists. If one does exist, it uses that one for the new connection. If one does not exist, it starts a new session for the user.

Calling the **Logout** method closes both the connection to the TIBCO iProcess Objects Server and the SAL session for the user. This is what was done in thick-client applications. However, in a stateless environment, this is NOT what you want to do, because the next time the user connects, a new SAL session must be started for the user, putting unneeded load on the TIBCO iProcess Objects Server, SAL, login daemon, and the Work Queue Server.

In a stateless environment, use the **SWEntUser.Disconnect** method. This method *disconnects* the user from the TIBCO iProcess Objects Server, leaving the user's SAL state present in the TIBCO iProcess Objects Server. When the user returns to the web page, a login is performed (with the Login method). The TIBCO iProcess Objects Server notices that there is already a SAL session open for that user, so it doesn't start one, making the connection very fast.

When the user is completely done and does not expect to issue any more requests to the TIBCO iProcess Objects Server, the **Logout** method should be called to clear the now unneeded SAL session in the TIBCO iProcess Objects Server.

Stateless Objects

When building web-based applications, maintaining the state of objects between pages is not efficient because of the stateless nature of the web. If you use TIBCO iProcess Objects in your web-based application, you would normally be required to reconstruct the objects, through the hierarchy of the object model each time a page is accessed. This results in a large volume of network traffic between the server and client, and a high latency that is not acceptable in a web environment. Therefore, to facilitate the building of web-based applications, methods are provided that allow you to create *stateless* objects.

The methods used to create stateless objects provide direct access to the objects residing in lists within the hierarchy of the object model, without requiring you to traverse object lists and create the underlying objects in the object model hierarchy. This minimizes the volume of data passed between the client and server.

A subset of the objects in the object model are suitable for stateless processing. Some objects, such as SWForm, SWMarking, and SWPriority, have no meaning when used outside of the object hierarchy. For example, SWMarking, which represents a field placed on a Staffware form, is not useful when created by itself. It is only useful when it is derived from the SWStep object to which it belongs.

The following table lists the objects that *are* suitable for use in a stateless environment, as well as the names of the methods used to create those objects without traversing the object model hierarchy.

Object	Stateless Method
SWAttribute	MakeAttribute
SWAWorkQ	MakeAWorkQ
SWCase	MakeCase
SWGGroup	MakeGroup
SWListValidation	MakeListValidation
SWNodeInfo	MakeNodeInfo
SWOSUser	MakeOSUser
SWProc	MakeProc
SWRole	MakeRole
SWStep	MakeStep

Object	Stateless Method
SWTable	MakeTable
SWUser	MakeUser
SWWorkItem	MakeWorkItem / MakeWorkItemEx
SWWorkQ	MakeWorkQ

The primary purpose of these methods is to provide a way for a web-based application to access an object without having to reconstruct it through the object hierarchy each time a new web page is accessed. It is also important to note that using these methods is a much more efficient way to access an object if the list containing the object on the server contains a large number of objects — this prevents having to download the entire list from the server.

Single-Parameter Methods

The stateless methods listed in the table above all require parameters that provide information about the object that had already been created on a previous web page. For example, if you were to create an **SWProc** object with the **MakeProc** method, you need to provide the name of the procedure and, optionally, the name of the node:

```
MakeProc(ProcName, [NodeName])
```

Using this method would require you to have saved the procedure name and node name somewhere so that they could be used on a future web page to re-create the SWProc object.

To make this process easier, *single-parameter* methods are provided. Single-parameter methods are available to re-create the stateless objects that require more than one parameter. These are:

- **MakeAWorkQByTag(Tag)**
- **MakeCaseByTag(Tag)**
- **MakeNodeInfoByTag(Tag)**
- **MakeProcByTag(Tag)**
- **MakeStepByTag(Tag)**
- **MakeWorkItemByTag(Tag)**
- **MakeWorkItemByTagEx(Tag, [CDQPNames], [CaseFieldNames])**
- **MakeWorkQByTag(Tag)**
- **MakeViewItemsByTag(Tag)**

Each of these methods requires only a single *Tag* parameter that consists of a string containing all of the information needed to re-create that object.

Using the Tag Property

Each of the objects that can be re-created with the stateless single-parameter methods, also has a **Tag** property that can be used to re-create that object. This property contains the information needed to re-create the object at a later time in a stateless environment.

For example, the **SWNodeInfo.Tag** property contains the computer name, node name, IP address, and TCP Port number for the SWNodeInfo object. That's all the information needed to re-create that object in a stateless environment.

Typically, you would save the Tag property from the desired object, then use it as an argument for a 'ByTag' method to create the desired object at a later time (e.g., a subsequent web page).

The Tag property allows you to create a stateless object by maintaining a single value rather than maintaining multiple values across web pages. See the example below.

```
for each oNodeInfo in oEnterprise.NodeInfos
  if Request.Form("Server") = oNodeInfo.SWEOSrvDesc then
    'found the matching SWNodeInfo object
    session("NodeTag") = oNodeInfo.Tag 'save Tag property
    session("NodeDesc") = oNodeInfo.SWEOSrvDesc 'save SWEOSrvDesc property
    exit for
  end if
next
```

Then, on a subsequent web page, the Tag property information that was saved can be used to re-create that object. For example:

```
set oEnterprise = CreateObject("SWECom.SWEnterprise.1")
oEnterprise.IsBroadcast = false 'disable UDP broadcast for servers, will use
                                'MakeNodeInfoByTag to save time on login

set oNodeInfo = oEnterprise.MakeNodeInfoByTag(session("NodeTag"))

set oEntUser = oEnterprise.CreateEntUsers(session("SWUserName"))
.
.
.
```

Passing an Empty Object in the Make Methods (C++ only)

When using TIBCO iProcess Objects (C++), the "make" methods require that you create a new "empty" object, then pass that object as an argument with the method call (the **makeNodeInfo** and **makeNodeInfoByTag** methods are an exception -- they are used to add a node to the **NodeInfos** list). For example:

```
SWGroup *makeGroup(SWGroup *pSWGroup,
                  char *GroupName);
```

*Note - The requirement to pass a new empty object has been extended to a few "non-make" methods as well, specifically, the **getXList**, **getXListPredict**, and **getExtWorkItem** methods.*

This is required because of memory allocation rules; TIBCO iProcess Objects use a memory management tool (Microquill's SmartHeap) that requires, by license agreement, that only TIBCO iProcess Objects use its heap. For this reason, the application must use the normal heap instead of the Smart-Heap heap. Therefore, the application must create the empty object so that it is on the normal heap.

After the application is finished using the object that it is "making", it must explicitly delete that object to prevent a memory leak. An example is shown below. This example uses the makeGroup method -- all of the other "make" methods work similarly.

```

SWEnterprise *pSWEnterprise;
SWEntUser *pSWEntUser;
SWNode *pSWNode;
SWList *pSWGGroups;
SWGGroup *pSWGGroup;
SWGGroup *pSWGGroup1;

try {
    pSWEnterprise = new SWEnterprise;
    pSWEntUser = pSWEnterprise->createEntUser(EntUserName);
    pSWEnterprise->setBroadcast(false);
    pSWEnterprise->addNode(ServerName, NodeName);
    pSWNode = pSWEntUser->login(NodeKey, Password, UserName);
    pSWGGroups = pSWNode->getGroups();
    pSWGGroup = (SWGGroup *) pSWGGroups->item(0);
    pSWGGroup1 = pSWNode->makeGroup(new SWGroup(), pSWGGroup->getName());
    // This newly created SWGroup object...
    ... (use pSWGGroup1 object) ...
    delete pSWGGroup1; // ...must be deleted to prevent leak.
    pSWEntUser->logout(NodeKey);
    delete pSWEnterprise;
}
catch (SWException Except) {
    LOG(CTLOGALL, "???", "
        "Number %d Description '%s'\n",
        Except.Number, Except.Description);
}

```

Because of the memory management tools used by TIBCO iProcess Objects, the following rule must be followed to prevent memory leaks: if TIBCO iProcess Objects create an object, they will delete it; if the client application creates an object, it must delete it.

Persisted XLists

The client can ask the TIBCO iProcess Objects Server to persist a collection of work items. (Only work items can be persisted.) This allows the client to disconnect from the server, then reconnect at a later time and have access to the same collection of work items from the previous connection. The most obvious use of this feature is for web-based applications to provide access to a consistent list of work items for a given user between successive web pages.

For information about persisting XLists, see [“Working with Persisted XLists” on page 81](#) in the [“Working with Lists”](#) chapter.

16

Optimizing Your Applications

Introduction

By their nature, object models provide a lot of flexibility in how you program with them. With the TIBCO iProcess Objects object model, however, there are a number of areas in which you can realize improved efficiency by performing functions in a certain way.

This chapter outlines those areas in which you can optimize your client applications.

Handling Large Lists of Work Items, Cases, Users, OS Users, Groups

When the object model was first created, the **SWView** object was the list object that was used to store filtered and sorted lists of work items and cases. And the **SWList** object was used to store lists of users, OS users, and groups.

It was determined later that when working with views and lists containing very large numbers of items (in the multiple thousands), they were not as efficient as desired. When you access an item for the first time in a view or list (or rebuild the list), all of the available items are retrieved from the server regardless of whether or not the application needs all of the items. Furthermore, to get the last item in the raw data buffers requires sequentially parsing all raw data into objects. Depending on the number of items, this can take a considerable amount of time. That's why the **SWXList** object was conceived.

“XLists” are more efficient than SWViews or SWLists because they only download a “block” of items from the server instead of *all* the available items. This makes for a much faster access when dealing with very large lists, especially in situations when you only need to access a few of the items in the large list.

All new development should use SWXLists whenever possible to make the application more efficient.

See [“SWXLists” on page 73](#) for detailed information about using XLists.

Clear Blocks on Client when using XLists

The blocks of work items retrieved from the TIBCO iProcess Objects Server and maintained in the XList consume local memory. The following are provided to control this memory usage:

- **IsKeepLocalItems** property - This read/write property on SWXList allows you to specify whether or not multiple blocks of items should be kept locally as subsequent blocks are retrieved from the server. This allows you to conserve local memory by specifying that only one block at a time be held locally. If set to True, multiple blocks will be kept locally. If set to False (the default), the previous block will be removed from the XList when a new one is retrieved from the server, i.e., only one block is kept locally at a time.
- **Clear** method - This method on SWXList frees memory and removes all items from the list on the client. Note that when called from an SWXList (this method is also available on other types of lists), it clears only the list on the client, not the buffers. Which means that if you clear an XList with the Clear method, then access an item, you are getting the same buffer data as before you cleared the list.

See [“Populating an XList of Work Items” on page 77](#) for more information about how blocks of work items are retrieved from the server.

Optimizing Communications between Client and Server

Below is a list of the actions you can take to minimize the overhead involved in communicating between the client and server:

- In a thick-client application, maximize the re-use of objects. To do this, don’t continually log in and out, which causes the SWNode object and all of its dependent objects to be destroyed.
- Use filtering to retrieve only the items that the application is specifically interested in. This reduces network traffic by causing only the items that match the filter criteria to be retrieved from the server.
- If using SWViews, use the **MaxCnt** property to limit the number of items to retrieve from the TIBCO iProcess Objects Server.
- Only use the “alternate” view/XList methods (**MakeViewItems**, **MakeXListItems**, **MakeViewCases**, **MakeXListCases**) when you need to view work items across multiple work queues or nodes at one time. Whenever possible, use the “default” view/XList (**WorkItems**, **WorkItemsX**, **Cases**, and **CasesX**).
- Don’t unnecessarily call the **Rebuild** method. This can potentially cause large amounts of unnecessary network traffic. Rebuild should be called only when you know that the list is out of date.

Other than administrative- or reporting-type applications, Rebuild should only be done on SWViews or SWXLists of SWCases and SWWorkItems.

- When working with web-based applications, use the available *stateless* objects to prevent having to create all of the underlying objects, resulting in less network traffic between the server and client. See [“Stateless Programming” on page 270](#) for information about stateless objects.

Filtering and Sorting in an Efficient Manner

The way in which you specify your filter and sort criteria can have a significant impact on how efficiently the filter and sort operation is performed. Flow diagrams are provided in the *Filtering Work Items and Cases* chapters that illustrate how to make your filter and sort operation more efficient. See the *Filtering/Sorting in an Efficient Manner* sections on [page 103](#), [page 131](#), or [page 157](#) (depending on which of the filtering enhancements have been implemented in your TIBCO iProcess Objects Server) for a description of each of the flow diagrams.

How Getting Case Data affects Application Efficiency

The *Filtering Work Items and Cases* chapters discuss the impact of getting case data when filtering and sorting. You should understand, however, that there is a performance impact on your application *anytime* you get case data.

The **CaseFieldNames** property contains the names of the case's fields that will be returned in the **Fields** property of SWCase. If this property is empty (the default), no fields are returned. Developers can add and remove field names to/from the CaseFieldNames property, or add a value of **&ALL&** to specify that *all* fields be returned.

The easy thing to do is to pass **&ALL&** to CaseFieldNames to get all of the fields. This can adversely affect performance, however. Don't use **&ALL&** unless you really do need all or most of the fields in the case(s).

Getting Case Data on View/XList vs. Case

The CaseFieldNames property is on SWCase, as well as SWView and SWXList (it's actually on SWCriteriaC and SWCriteriaWI, which are used to specify criteria for the SWXList).

If you need case data for all or most of the cases in a view or XList, it is much more efficient to specify the field names in the CaseFieldNames property on the SWView or SWCriteriaC/SWCriteriaWI object. This causes the TIBCO iProcess Objects Server to return the fields for each case in the view or XList in a single request to the server.

However, if you need case data for only a few cases in the view or XList, it is more efficient to specify the field names on each specific SWCase object. This causes the client to make separate requests for each case on which you need the case data.

Looping Through Items in an SWList or SWView

This section describes the proper way of looping through items in an SWList or SWView.

Note - If you are dealing with very large lists, you should be using SWXLists instead of SWViews or SWLists. See “Handling Large Lists of Work Items, Cases, Users, OS Users, Groups” on page 275 for information about using SWXLists.

If you want to loop through all items in an SWView, DO NOT loop on the **Count** property. Instead loop on the **IsEOL** property, accessing items sequentially until the IsEOL property is true.

```
oWorkQ.WorkItems.MaxCnt = 20
oWorkQ.WorkItems.IsEOL = False
While Not (oWorkQ.WorkItems.IsEOL)
    Set oWorkItem = oWorkQ.WorkItems.Item(i)
    .
    .
    i = i + 1
Wend
cnt = oStartProcs.Count
```

Note - It is recommended you work on a small set of items at a time by setting MaxCnt. A reasonable value is 20-50.

Or use the **For Each** construct in VB:

```
For Each oWorkItem In oWorkQ.WorkItems
    ListCtl.AddItem oWorkItem.Key
Next
```

The **Count** property reflects the number of objects that have been created and placed on the list/view at the client, not necessarily all objects that are available from the server.

The **IsEOL** property will equal true when the last object available in the buffers has been created into an object and placed on the list/view. At that point, the Count property will accurately reflect the number of items available on the server.

Locking, Keeping, Releasing Multiple Items

If you are going to be processing multiple work items that are on an SWView or SWXList, it is much more efficient to use the methods on the SWWorkQ object. They are:

- **LockItems**
- **KeepItems**
- **ReleaseItems**
- **UnlockItems**
- **UndoItems**
- **LockItemsEx**
- **KeepItemsEx**
- **ReleaseItemsEx**
- **UnlockItemsEx**
- **UndoItemsEx**

These methods allow the client to process multiple work items with a single message trip to the server. Whereas, if you are using the singular version of these methods (LockItem, KeepItem, etc.) on the SWWorkItem object, a message is sent to the server for each individual work item processed.

Optimizing VB Application Performance

If you are developing in Visual Basic, use early binding (i.e., reference the SWECom.dll in the VB project and use the “new” operator to create objects) instead of late binding (which uses the CreateObject operator to create objects) when instantiating objects. For example:

```
Set oEnterprise = New SWEnterprise
```

Note that there are only three objects that can be created by the programmer — SWEnterprise, SWSortField, and SWMarking. All others are instantiated by other objects. (The TIBCO iProcess Objects (Java and C++) also have an SWDate object that can be created by the programmer.)

Accessing a Single Object

If you need to access a single object (work queue, work item, case, etc.), and you know which object you need, it is much more efficient to use one of the “Make” methods (**MakeWorkQ**, **MakeCase**, **MakeWorkItem**, etc.) These methods allow you to provide the name of the desired object, and have that object directly created. The only requirement is that you know the name of the object you need created.

Using the Make methods significantly reduces network traffic by eliminating the need to reconstruct the entire object hierarchy to get to the single desired object.

See “[Stateless Objects](#)” on [page 271](#) for more information about these methods.

Clear Unneeded Views and XLists

When you are finished using SWViews and SWXList in your client application, clear them to free memory and system resources:

```
oWorkQ.WorkItems.Clear
```

Case Indexing

Lists of cases are indexed by case number (SW_CASENUM) and case reference number (SW_CASEREF). Therefore, if you need to search a list of cases for one case or a small number of cases, searching on the case number or case reference number is very fast. For example:

```
SW_CASENUM = 5  
or  
SW_CASEREF="2-6"
```

Note - Case number is an integer; case reference number is a text string.

Searching a list of cases using any other expression causes the TIBCO iProcess Objects Server to have to search through the entire list, adversely affecting performance.

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 280](#).
- **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 290](#).
- **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 291](#).

Client Log

The client log records messages generated from the client application. These messages are most useful for debugging.

Note - The following logs are also available as noted:

- **TIBCO iProcess Objects Server Log** - This log records messages generated by the TIBCO iProcess Objects Server. 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*.

When the client needs 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 (these were formally written to the Application Event Log on NT). So if no errors are occurring, no log is created.

The client log contains one line for each message. They are in the format:

```
f|ppppp|tttt|hhhhhhh|dd/dd/dddd dd:dd:dd.ddd|www|PPPPPPPP|ccccccc|mmmmmmmm|llll|UserMsg
```

where:

f	Line format (B - Basic, M - Contains memory values)
ppppp	Process ID
ttttt	Thread ID
hhhhhhh	Hood ID, where: 00000000 = Msg from SWLog 00000001 = Msg from receive thread 00000002 = Msg from SWEnterprise initialization 00000003 = User-created object (SWMarking, SWSortField) 00000004 = JNI boundary error 00000009 = Undefined hood 0000000A = After SWEnterprise created, but before login (SWNode) 0000000B & above = Logged in user node returns
dd/dd/ddd dd:dd:dd.ddd	Date and time
wwwwwwwww	Working set size (only appears if IsShowMemory = True)
PPPPPPPPP	Page file usage (only appears if IsShowMemory = True)
cccccccc	Log category (hex format)
llll	Log level (ERR, WARN, INFO, DEBUG)
mmmmmmm	Message type (see SWLogMessageType on page 287)
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 SWLog hood (00000000).

Logging on to a server (which creates a new SWNode object) generates a new Hood ID. All objects in the hierarchy below this node are in the same hood (0000000B and above).

Objects in the hierarchy prior to login (like SWEnterprise, SWEntUser, etc.) each have the Hood ID of SWEnterprise initialization.

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|00076|0015C|00000001|06/05/2001 08:27:43.752|00000100|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

You can dynamically control the client log. It can be turned on and off, its location can be modified, you can specify the types and categories of messages to log, etc.

You can control the client log using the following methods:

- using the **SWLog** object
- through **Registry settings** (Windows only)
- through **environment variables** (UNIX only)

These are described in the following subsections.

The SWLog Object

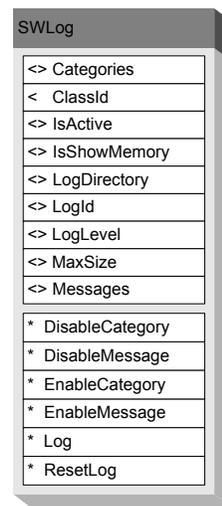
The SWLog object allows configuration and control of the client log from the client application. It contains properties and methods that can be used to define the location of the log file, the level of logging, the size of the log, etc.

The SWLog object is created when the SWEnterprise object is instantiated. It is accessible from the **SWEnterprise.ClientLog** property.

Note - The SWEnterprise object also contains a `ReceiveLog` property that has been deprecated. New code should use the `ClientLog` property. The `ReceiveLog` property will return the same SWLog object as the `ClientLog` property.

The functions provided through the properties and methods of the SWLog object are described in the subsections that follow.

All objects share the same log, so if there is more than one SWEnterprise object in the client application, changing SWLog of one SWEnterprise also effects the other.



Registry Settings (Windows only)

When the SWLog object is initially created (upon SWEnterprise creation), it attempts to obtain its property values from the Registry. If the Registry entries exist, they are written to the properties of the SWLog object. If the Registry entries do not exist, a Registry key is created at the path shown below.

```
HKEY_CURRENT_USER
  Software
    Staffware plc
      Staffware SEO Client
        Logs
          SWClient_VB6
```

Note - The Registry path is slightly different if running from IIS. Instead of HKEY_CURRENT_USER, the Registry location is HKEY_USERS\DEFAULT.

When the Registry entries are initially created, the name of the key is determined by the value of the **LogId** property of the SWLog object. In Windows, the default LogId value is the name of the procedure in which the SWLog object is created, prefaced with “**SWClient_**”. For example, if the SWLog object is generated from the Visual Basic 6 IDE environment, the Registry key will be “**SWClient_VB6**”.

Since the SWLog object 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 SWLog 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).

When the Registry entries are created, the default values of the SWLog object are written into the Registry. They are:

- Active = “1” (true)
- Categories = “7FFFFFF3” (all categories except swCatConstDestr and swCatReceiveThread)
- LogDirectory = “C:\TEMP\”
- LogLevel = “1” (Error level — the lowest level)
- MaxSize = “15”
- Messages = “7FFFFFFF” (All messages)
- ShowMemory = “0” (No)

After the SWLog object is created, if a property value is changed, that change is written to the Registry and it becomes active.

Changing the **LogId** property of the SWLog object causes a new Registry key to be created and initialized with the current values of the SWLog properties. Any SWLog property changes made after changing the LogId causes the change to be written to the Registry under the new LogId.

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 \(Windows only\)](#)” above are used.

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 (SEOClient_Active environment variable) is set to “0” (false).

The following are the UNIX environment variables that can be created to control the client log:

- SEOClient_Active
- SEOClient_Categories
- SEOClient_LogDirectory
- SEOClient_LogLevel
- SEOClient_MaxSize
- SEOClient_Messages
- SEOClient_ShowMemory

Name and Location of the Client Log

Log File Name

The **LogId** property on the SWLog object specifies the name of the client log (as well as the Registry key name). If you change this property, a new client log is created with the new name (a new Registry key is also created with the new name).

The default client log name depends on the operating system you are using, as follows:

- **Windows** - The name of the procedure in which the SWLog object is created, prefaced with “**SWClient_**”. For example, if the SWLog object is generated from the Visual Basic 6 IDE environment, the Registry key will be “**SWClient_VB6**”.
- **UNIX** - The name defaults to “**SEOClient**”.

Log File Directory

The client log will be written to the directory specified in the **LogDirectory** property of the SWLog object.

The default directory depends on the operating system you are using, as follows:

- **Windows 98, ME, and NT** - The value of the TEMP environment variable (typically, “**C:\TEMP**”).
- **Windows 2000 and XP** - Documents and Settings*username*\Local Settings\Temp\
- **UNIX** - The directory defaults to “**/tmp/**”.

Activating / Deactivating the Client Log

The **SWLog.IsActive** property specifies whether or not messages are written to the client log. This includes “always log” messages (category swCatAlwaysLog). This property allows you to turn off logging without having to clear the category and message filter settings (which are described in the next section). The default setting for IsActive is true — logging is enabled.

Setting IsActive to False does not inhibit writing the SWLog properties to the Registry; it merely causes all messages to be filtered out.

You can use the `IsActive` property to prevent the client log from ever being created. Before the `SWLog` object is created (prior to instantiating `SWEnterprise`), set the **IsActive** flag in the Registry to “0” (false). When the `SWLog` object is created, it will obtain its default values from the Registry, including setting the `IsActive` property to false (based on the setting of the `Active` flag in the Registry).

Filtering the Client Log

The `SWLog` object contains a number of properties and methods that allow you to specify the categories and types of messages that are written to the client log. The subsections that follow 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 `SWLog.LogLevel` property. The `SWLogLevelType` enumerations can be used to specify the amount of information to record:

SWLogLevelType	Value	Amount of Information
swLogError	1	Least
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.

The default log level is `swLogError`.

Note - Is is not recommended that you select `swLogDebug` along with `swCatAll`, as the amount of log information can impact performance.

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 `SWLog.Categories` property. The `SWLogCategoryType` enumerations can be used to specify the categories you would like written to the log.

SWLogCategoryType	Value
swCatAll	0x7FFFFFFF
swCatAlwaysLog	0x00000001
swCatSEouser	0x00000002
swCatConstDestr	0x00000004
swCatReceiveThread	0x00000008
swCatMessages	0x00000010
swCatMsgSend	0x00000020

SWLogCategoryType	Value
swCatMsgReceive	0x00000040
swCatUDP	0x00000080
swCatWinSock	0x00000100
swCatConversion	0x00000200
swCatTiming	0x00000400
swCatMethodCalls	0x00000800
swCatObjectWrapping	0x00001000
swCatMemory	0x00002000

Notice that a unique bit is set for each category, allowing you to specify multiple categories with a single designation. Set the **SWLog.Categories** property to the hex value that is the combined value for all the categories you want written to the client log.

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 **SWLog.Categories** property 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 SWLog 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 the client sends messages to the TIBCO iProcess Objects Server. This is done by using the **SWLog.Messages** property. The **SWLogMessageType** enumerations can be used to specify the message types that you would like written to the log. (Note that this is applicable only if the swCatMessages category is enabled; see the previous section.)

Notice that a unique bit is set for each message type, allowing you to specify multiple message types with a single designation. Set the **SWLog.Messages** property to the hex value that is the combined value for all the message types you want written to the client log.

The default category is swMsgAll — all message types are written to the log.

Enabling and Disabling Messages

The SWLog 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 **SWLogMessageType** 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.

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

Displaying Memory Information in the Client Log (Windows only)

You can display memory information in the client log by setting the **SWLog.IsShowMemory** property to true. This causes the **Working Set Size** and **Page File Usage** parameters to appear in each log entry (see the example below).

```
BM|00129|0008A|0000000B|07/02/2001 09:15:59.942|022851584|013824000|00000004|7FFFFFFF|.....
                                     |-----|-----|
                                     Working   Page
                                     Set       File
                                     Size      Usage
```

Working Set Size is the current number of bytes in the Working Set of this process. The Working Set is the set of memory pages touched recently by the threads in the process. If free memory in the computer is above a threshold, pages are left in the Working Set of a process even if they are not in use. When free memory falls below a threshold, pages are trimmed from Working Sets. If they are needed, they will be soft-faulted back into the Working Set before they leave main memory.

Page File Usage is the current number of bytes this process has used in the paging file(s). Paging files are used to store pages of memory used by the process that are not contained in other files. Paging files are shared by all processes, and lack of space in paging files can prevent other processes from allocating memory.

Adding Entries to the Client Log

The **SWLog.Log** method allows you to add text messages to the client log from your application. For example:

```
oLog.Log "This is displayed in the client log"
```

Causes the following message to appear in the client log:

```
.... 13:35:19.946|00000002|7FFFFFFF|INFO|SWLog.Log: This is displayed in the client log
```

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 **SWLog.MaxSize** property. 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 **SWLog.ResetLog** method can be used to clear the client log. After clearing all existing entries, an initial header message is written to the log.

Testing the Client Log

The SWLog object contains a hidden method that allows you to test the client log. The **LogTest(*TestId*)** method writes test messages to the client log based on the *TestId* parameter specified:

TestId	Test Message	Description
1	Category/Level	One log message of each category type at each log level.
2	Message	One log message of each message type at the error level.
3	Multiple Category	Log messages with multiple category bits on a single log message.
4	Multiple Message	Log messages with multiple message bits on a single log message.
5	Always Log	A single log message at the error level with the always log category bit set.

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 want 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.

The following is the Registry DWORD value that must be added if using Windows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Staffware plc\Staffware SEO Client\MessageWaitTime
```

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 “MsgSegWaitTime(*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.

— `TISOUUnicodeConverterName`

On Windows systems, the Registry entry must be located in the following path:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Staffware plc\Staffware SEO Client\
```

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 `TISOUUnicodeConverterName` 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 `TISOUUnicodeConverterName` environment variable does not exist or is set to an invalid converter name.

Windows Systems

If the `TISOUUnicodeConverterName` 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 SEO Client\
```

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 `TISOUUnicodeConverterName` Registry entry does not exist or is set to an invalid converter name.

For more information about ICU, see:

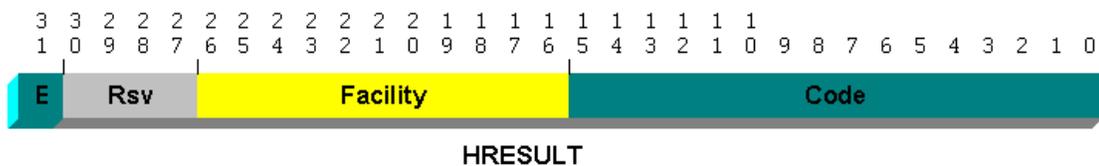
<http://icu.sourceforge.net/>

19

Error Handling

TIBCO iProcess Objects (COM)

All errors from TIBCO iProcess Objects (COM) are returned as exceptions. The exceptions are returned in the standard HRESULT specified by COM as follows:



where:

E - The severity code

0 - Success

1 - Error

Rsv - Reserved bits

Facility - Facility code. This code will be FACILITY_ITF(4) for errors generated by TIBCO iProcess Objects (COM).

Code - The error code generated by the facility. You can determine whether the TIBCO iProcess Objects Server or the client generated the error as follows:

TIBCO iProcess Objects Server - Error code value is less than 0x800.

Client - Error code value is greater than or equal to 0x800.

For a complete list of the client and Server error codes and messages, see the on-line help system.

Error Constants

The TIBCO iProcess Objects (COM) provides enumeration type constants that represent each of the errors that are generated by the client and server. They are:

- **SWClientErrorType** - Errors generated by the client
- **SWServerErrorType** - Errors generated by the TIBCO iProcess Objects Server

These constants can be used in code instead of a number, providing more readable code. The example shown on the next page shows how these constants can be used in error handling in VB. See the on-line help for a complete list of the client and server error constants.

Error Trapping in Visual Basic

When using TIBCO iProcess Objects in Visual Basic, it is important to trap errors using exception handling with the appropriate **On Error** function, otherwise run-time errors in VB will be "fatal" and execution of the application may halt, which could mean the loss of unsaved work. This could mean that data entered into a work item could be lost.

It is preferable to try to use the Visual Basic error handler where possible to either warn the user of a problem, exit gracefully or provide an alternative action that may not cause the error. To do this, use the **On Error** function in each sub procedure or function that contains methods or properties. The example below shows using the **Err** object to look into the cause of the problem and if appropriate rectify it before continuing.

```
Sub TWorkQ(oTWorkQ As swworkq)
    Dim IsReadOnly As Boolean

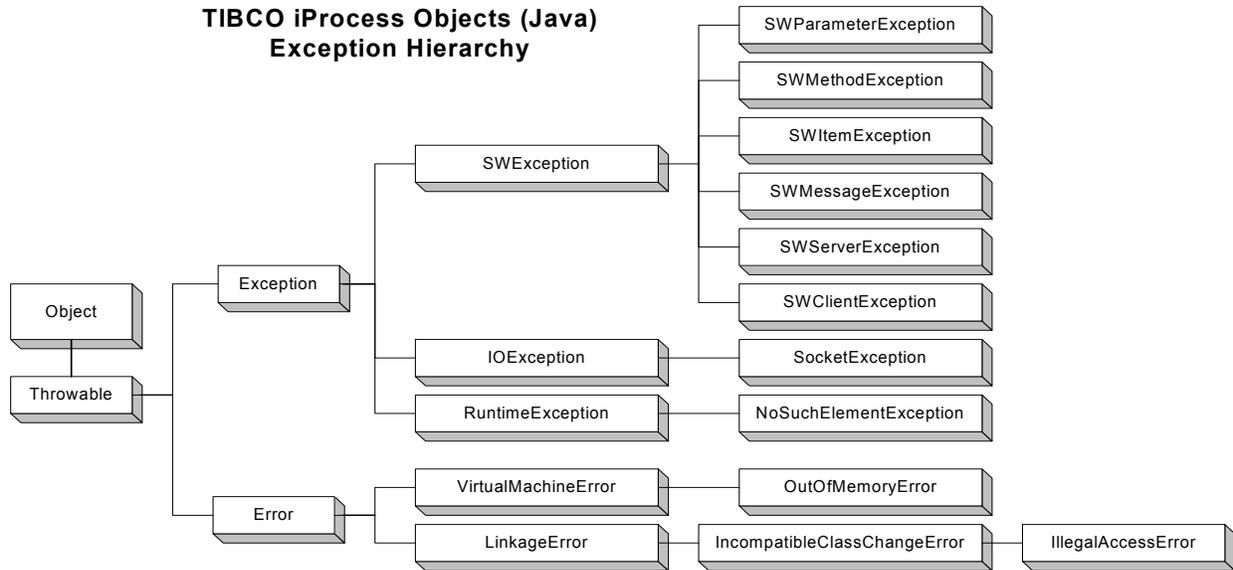
    On Error GoTo WorkQError
    IsReadOnly = oTWorkQ.IsReadOnly
    Exit Sub

WorkQError:
    If Err.Number = vbObjectError + swNoQueueErr _
        Or Err.Number = vbObjectError + swNoMemberErr _
        Or Err.Number = vbObjectError + swNotAuthErr Then
        'oEnterprise is global variable
        oEnterprise.ClientLog.Log ("TWorkQ: exception, no permission for _
            isReadOnly," _
            & " Number " & Err.Number - vbObjectError _
            & " Description " & Err.Description & vbCrLf)
    Else
        oEnterprise.ClientLog.Log ("TWorkQ: unexpected exception, isReadOnly," _
            & " Number " & Err.Number - vbObjectErrors _
            & " Description " & Err.Description & vbCrLf)
    End If
End Sub
```

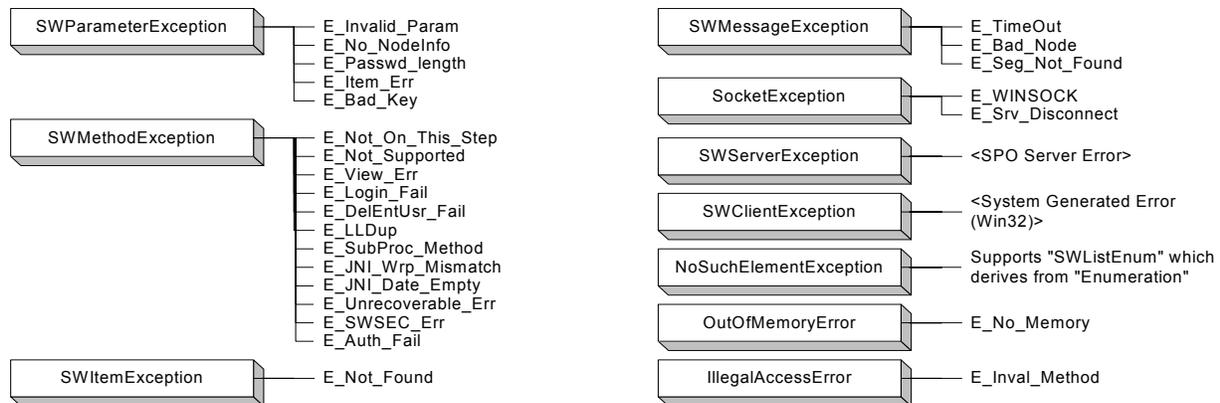
Notice that when comparing the value of an error constant with the `Err.Number` value, you must also add the VB error constant, **vbObjectError**, to the error constant.

TIBCO iProcess Objects (Java)

All errors are returned from TIBCO iProcess Objects (Java) via exceptions, as indicated by the exception hierarchy shown below:



Errors to Exception Assignment



Error Files

err.h
 <TIBCO iProcess Objects Server Error>
 SPO Server generated errors and associated error strings.

EOErrs.h
 TIBCO iProcess Objects (Java)-generated errors and associated error strings. Routines to return error string with length checking.

SWErrors.cpp
 Java: jThrowError - Mapping of errors to Java Exception Objects.
 COM: SetErrStr - SetHResStr -

Error Constants

The TIBCO iProcess Objects (Java) provides enumeration type constants that represent each of the errors that are generated by the client and server. They are:

- **SWClientErrorType** - Errors generated by the client
- **SWServerErrorType** - Errors generated by the TIBCO iProcess Objects Server

These enumeration constants can be used in code instead of a number, providing more readable code.

The example below shows how the error constants are used in error handling in a Java Client application.

```
void tWorkQ(SWWorkQ oTWorkQ) throws
    SWServerException, SWClientException, SWMethodException, SWItemException,
    SWParameterException, SWMessageException, java.net.SocketException{
try{
    boolean isReadOnly;
    isReadOnly = oTWorkQ.isReadOnly();
}
catch(SWServerException err){
    if (err.getErrCode() == SWServerErrorType._swNoQueueErr _
        || err.getErrCode() == SWServerErrorType._swNoMemberErr _
        || err.getErrCode() == SWServerErrorType._swNotAuthErr) {
        // oEnterprise is class variable
        oEnterprise.getClientLog().log ("TWorkQ: exception, no permission _
            for isReadOnly," _
            + " Number " + err.getErrCode() _
            + " Description " + err.getMessage());
    }
    else {
        oEnterprise.getClientLog().log ("TWorkQ: unexpected exception, _
            isReadOnly," _
            + " Number " + err.getErrCode() _
            + " Description " + err.getMessage());
    }
}
}
```

For a complete list of the client and Server error codes and messages, see the on-line help system.

TIBCO iProcess Objects (C++)

When using TIBCO iProcess Objects (C++), all errors are written to the **SWException** object. This object's **Number** and **Description** properties are available to view and act upon the error.

The C++ Client includes constants that can be used in your code instead of the actual raw error number that is generated, which doesn't have as much meaning when reading the code. The C++ constants are actually #defines from the **EOErrs.h** (client errors) and **err.h** (server errors) files (see **ER_NOQUEUE**, etc., in the example below).

```
TWorkQ(SWWorkQ * pSWWorkQ {
    try {
        isReadOnly = pSWWorkQ->isReadOnly();
        LOG(CTLOGALL, "TWorkQ: isReadOnly %d\n", isReadOnly);
    }
    catch (SWException Except) {
        if (Except.Number == -(ER_NOQUEUE)
            || Except.Number == -(ER_NOMEMBER)
            || Except.Number == -(ER_NOTAUTH)) {
            LOG(CTLOGALL, "TWorkQ: exception, no permission for isReadOnly, "
                "Number %d Description '%s'\n",
                Except.Number, Except.Description);
        } else {
            LOG(CTLOGALL, "TWorkQ: unexpected exception, isReadOnly, "
                "Number %d Description '%s'\n",
                Except.Number, Except.Description);
            assert(false);
        }
    }
}
```

For a complete list of the constants available for client and Server errors, see the on-line help.

Handling Multiple Work Item Operation Errors

The following methods allow you to perform operations on multiple work items on a work queue with a single method call:

- LockItems
- UnlockItems
- ReleaseItems
- KeepItems
- UndoItems

When these methods are called, there is the possibility that one or more of the items in the array will return an error. If this occurs, instead of failing the entire operation, an error is written to the **SWWorkItem.LastError** property for the item(s) that had the error.

You must check the LastError property to determine which items failed. A value of zero means it was successful. See the on-line help system for a list of the errors that may be found in LastError.

Debugging Problems with ASP

TIBCO iProcess Objects can be used in a Web-based solution, where users have access to data on the TIBCO iProcess Engine via Microsoft's Internet Information Server (IIS) from a web browser using Active Server Pages (ASP) and possibly Visual Basic applications (helpers). In this scenario, when a problem arises between IIS and the TIBCO iProcess Engine, it may be difficult to investigate the cause of the problem, due to the complexity of the system. The IIS server may treat problems with the TIBCO iProcess Objects (COM) or other dynamic link libraries (i.e., the helpers) in the one of the following ways:

- IIS exits and a Dr. Watson error log is generated
- the Web service exits abnormally (IIS hangs)
- the browser returns an "ASP 0115" error
- the browser returns a "Server Application Error" message
- the browser returns a "Server Too Busy" error

To help investigate and debug such applications or environments where it is not immediately obvious which component of the system is causing the problem, Microsoft has created a tool called the **Exception Monitor** to monitor the IIS server. Information on how to troubleshoot with the IIS Exception Monitor can be found at:

<http://msdn.microsoft.com/workshop/server/iis/ixcptmon.asp>

The IIS Exception Monitor attaches to the IIS process and reports back what is going on. It can provide an explanation of which type of exception occurred, what thread the exception happened on, and what DLLs were executing functions on that thread.

This may not be enough information to solve the problem 100 percent of the time. However, it usually provides more information than you had before and narrows the scope of the problem down to focus on a certain ODBC driver or component that is being called.

Instructions on how to read the log files that are produced by the Exception Monitor can be found at:

<http://msdn.microsoft.com/workshop/server/iis/readlogs.asp>

To be able to debug Visual Basic DLLs properly you will need to compile a debug version of the DLL and export the symbols to a .pdb file. In addition, it may be necessary to use a debug version of the SWEOCOM.DLL while using the Exception Monitor. A debug version of the TIBCO iProcess Objects (COM) is included on the distribution CD. Contact your local TIBCO Support Representative if you need more information.

Error Messages

This section lists error messages that may appear, as well as probable causes and resolutions.

“An Exception of Type java.lang.UnsatisfiedLinkError was not handled”

Symptom

This error message can appear when attempting to run a java program from Microsoft's Visual J++ 6.0.

Cause

Some older versions of the Microsoft Visual J++ development tool may have a version of the Java Virtual Machine that does not support Sun's Java Native Interface (JNI), which is required to use TIBCO iProcess Objects (Java).

Resolution

Make sure that the Virtual Java Machine on your development platform supports both JNI and RNI.

“Authentication Request Failed”

Symptom

When performing a login using the SWEntUser.Login method, this error message is displayed.

Cause

Either an incorrect user name or password has been entered.

When logging a user into the TIBCO iProcess Engine, a sal_login is called only when no sessions for that user name already exist. The sal_login call will return an “Unknown username or bad password” error if it can't verify the user name and/or password. If a session already exists for the user, the user name and password are checked using a different function, which returns the “Authentication Request Failed” error if the password check fails.

Resolution

Ensure the correct user name and password are entered.

“Error 2140: An internal Windows NT error occurred”

Note - This is 1 of 5 of the same error message. To determine the specific cause, you must also look at the error message written to the TIBCO iProcess Objects Server log (SWEntObjSvXX.log). See the Symptom section.

Symptom

This error occurs when the TIBCO iProcess Objects Server service is started. The following error messages will also appear in the SWEntObjSvXX.log file:

```
281:186:09/17/1999 16:09:43.521:ERR:error calling OpenSWEntObjDB for SWAutoFwdQ (-167)
```

```
281:186:09/17/1999 16:09:43.521:ERR:error initializing autofwd/qview database (-167)
```

Cause

The number of message threads may be too high.

Resolution

Try decreasing the number of messages threads to 30 and restart the TIBCO iProcess Objects Server service. The number of message threads is configured using the TIBCO iProcess Objects Server Configuration Utility (for information on using this utility, see the *TIBCO iProcess Objects Server Administrator's Guide*).

“Error 2140: An internal Windows NT error occurred”

Note - This is 2 of 5 of the same error message. To determine the specific cause, you must also look at the error message written to the TIBCO iProcess Objects Server log (SWEntObjSvXX.log). See the Symptom section.

Symptom

This error occurs when attempting to start the TIBCO iProcess Objects Server service. You will also find one of the following two error messages in the SWEntObjSvXX.log file:

```
211:12b:06/16/1999 11:24:46.581:ERR:TCP bind() error (10048)
```

or

```
267:cd:07/06/1999 11:01:31.061:ERR:error looking up TCP service (seomkw)
```

Cause

You may be configuring your TIBCO iProcess Objects Server to use a specific TCP port number for client connections, rather than using "DEFAULT," which means use an ephemeral port (i.e., dynamic). If this is the case, there may be a problem with the configuration of the TCP service.

Resolution

To check whether you are using a specific or ephemeral port number, run the TIBCO iProcess Objects Server Configuration Utility and click on the TCP tab. If your TIBCO iProcess Objects Server is configured to use a specific TCP port for client connections, there will be a name in the TCP Port field other than "DEFAULT." If that is the case, you will need to make sure that the service name is set up in the Services file in the %System Root%\System32\drivers\etc directory and that the port number specified is valid. Be sure that the port you have chosen is not being used by another service or is too high.

You may want to specify DEFAULT to see if the TCP configuration is the problem.

See [“Configuring the TIBCO iProcess Objects Server TCP Port” on page 27](#) for more information.

“Error 2140: An internal Windows NT error occurred”

Note - This is 3 of 5 of the same error message. To determine the specific cause, you must also look at the error message written to the TIBCO iProcess Objects Server log (SWEntObjSvXX.log). See the Symptom section.

Symptom

This error occurs when attempting to start the TIBCO iProcess Objects Server service. The following error messages will appear in the SWEntObjSvXX.log file:

```
174:b4:05/10/1999 11:05:29.656:ERR:error in CoCreateInstance for SWAutoFwdQ (80070005)
```

```
174:b4:05/10/1999 11:05:29.656:ERR:error initializing autofwd/qview database (80070005)
```

Cause

If DCOM is enabled on your NT system, swpro (the background user) may not have the required Access, Launch, or Configuration Permissions.

Resolution

To check if DCOM is enabled:

1. Log on to Windows as an administrator.
2. From the task bar, choose **Start** and then **Run**.
3. Enter **dcomcnfg** and press **OK**. This runs the **Dcomcnfg.exe** configuration utility.
4. Click **Default Properties**.

If the check box labelled **Enable Distributed COM on this computer** is selected, DCOM is enabled and you must complete the following steps to give user **swpro** Access Permissions, Launch Permissions and Configuration Permissions.

To add permissions for swpro:

1. Click **Default Security**.
2. Set up Access Permissions. To do this:
 - i. In the **Default Access Permissions** frame, click **Edit Default** and then **Add**.
 - ii. In the **List Names From** list box, choose the appropriate computer or domain name where the TIBCO iProcess Engine is installed.
 - iii. Click **Show Users**.
 - iv. In the **Names** list, click the **swpro** user and then **Add**.
 - v. In the **Type of Access** list box, choose **Allow Access** and click **OK** to return to the **Registry Value Permissions** dialog.
 - vi. Click **OK** to return to the **Distributed DCOM Configuration Properties** dialog.
3. Set up Launch Permissions. To do this:
 - i. In the **Default Launch Permissions** frame, click **Edit Default** and then **Add**.
 - ii. In the **List Names From** list box, choose the appropriate computer or domain name where the TIBCO iProcess Engine is installed.
 - iii. Click **Show Users**.
 - iv. In the **Names** list, click the **swpro** user and then **Add**.
 - v. In the **Type of Access** list box, choose **Allow Launch** and click **OK** to return to the **Registry Value Permissions** dialog.
 - vi. Click **OK** to return to the **Distributed DCOM Configuration Properties** dialog.
4. Set up Configuration Permissions. To do this:
 - i. In the **Default Configuration Permissions** frame, click **Edit Default** and then **Add**.

- ii. In the **List Names From** list box, choose the appropriate computer or domain name where the TIBCO iProcess Engine is installed.
- iii. Click **Show Users**.
- iv. In the **Names** list, click the **swpro** user and then **Add**.
- v. In the **Type of Access** list box, choose **Full Control**.
- vi. Click **OK** three times to exit the DCOM configuration properties program.

“Error 2140: An internal Windows NT error occurred”

Note - This is 4 of 5 of the same error message. To determine the specific cause, you must also look at the error message written to the TIBCO iProcess Objects Server log (SWEntObjSvXX.log). See the Symptom section.

Symptom

This error occurs when attempting to start the TIBCO iProcess Objects Server service. The following error message will appear in the SWEntObjSvXX.log file:

```
259:125:06/15/1998 12:38:19.031:ERR:error calling OpenSWEntObjDB for SWAutoFwdQ (-2147417851)
```

Cause

The Microsoft Access Driver (.mdb) ODBC driver must be set up for the swpro user. Without this driver installed, the TIBCO iProcess Objects Server will fail to start.

Resolution

To check if you have the Microsoft Access Driver installed:

1. Log on to Windows as **swpro**.
2. From the **Control Panel**, choose the **Data Sources (ODBC)** icon.
3. Click the **Drivers** tab. If you have the driver installed, there will be an entry for **Microsoft Access Driver (.mdb)** in the list.

If you do not have the **Microsoft Access Driver** installed, you can install it from the TIBCO iProcess Objects Server CD. To do this:

1. From the TIBCO iProcess Objects Server CD, run **\\MicrosoftODBC\Mdacfull.exe**.
2. Follow the instructions presented on your screen.

“Error 2140: An internal Windows NT error occurred”

Note - This is 5 of 5 of the same error message. To determine the specific cause, you must also look at the error message written to the TIBCO iProcess Objects Server log (SWEntObjSvXX.log). See the Symptom section.

Symptom

This error occurs when attempting to start the TIBCO iProcess Objects Server service. The following error message will appear in the SWEntObjSvXX.log file:

```
219:a2:07/19/1999 14:34:32.081:ERR:Staffware multiple logins are disabled; must be enabled
```

Cause

The multiple logins feature is disabled — it must be enabled.

Resolution

Edit the *SWDIR\etc\staffpms* file and set the 13th character of line 4 to **Y** (where *SWDIR* is the directory where the TIBCO iProcess Engine is installed), then restart the TIBCO iProcess Objects Server Background service.

“Error 5: Access is denied”

Symptom

The TIBCO iProcess Objects Server service could not be started on Windows 2000.

Cause

The SYSTEM account does not have the proper permissions.

Resolution

Follow these steps:

1. Via Windows 2000 Explorer, right-click on the **SWDIR\bin\SWEntObjSv.exe** file, and select **Properties**.
2. On the **Security** tab, click **Advanced**. The **Access Control Settings** dialog is displayed.
3. On the **Permissions** tab, click **Add**. The **Select User, Computer, or Group** dialog is displayed.
4. Select **SYSTEM**, then click **OK**. The **Permission Entry** dialog is displayed.
5. Ensure that the **Allow** check box is checked for all of the permissions listed, then click **OK**.
6. Click **OK** again on the **Properties** dialog.

You should now be able to successfully start the TIBCO iProcess Objects Server service.

“Error Calling CreateDataSource Interface for SWAutoFwdQ”

“Error Initializing AutoFwd/QView Database”

Symptom

The TIBCO iProcess Objects Server on NT does not start correctly. The SwEntObjSvXX.log file contains the following error message:

```
00159|000AA|10/16/2000 10:07:40.718|00000008|ERROR|error calling CreateDataSource interface for SWAutoFwdQ (-175)
```

00159|000AA|10/16/2000 10:07:40.718|00000008|ERROR|error initializing autofwd/qview database (ffff51)

Cause

The error is being generated by the SWEntObjDB program. This program implements the COM interfaces that the TIBCO iProcess Objects Server uses to manipulate the SWEntObjDB.mdb database. The -175 error means that something at the lowest levels of ODBC or RDO has gone wrong. However, it is not possible for the TIBCO iProcess Objects Server to exactly pinpoint the cause of the problem.

One of the following issues could be causing the problem:

- Microsoft Access Driver (*.mdb) ODBC or Microsoft RDO 2.0 drivers (odbcjt32.dll and msrdo20.dll) are not installed correctly, or you have the wrong versions installed
- Version mismatch of system files
- Permission problems

Resolution

A test program that can help to determine the cause of this problem is available on the TIBCO Technical website. This program should be run by the **swpro** NT user account, and the appropriate path to the directory and nodename should be entered into the correct boxes on the form displayed. See DIN000237. (Contact TIBCO Technical Support if you don't have access to the TIBCO Technical website.)

“Error creating mutex”**Symptom**

Users are unable to log on. The SwEntObjSvXX.log file contains the following error message:

6045:13:12/06/2000 10:31:08.737:ERR:error creating mutex (SWENTOBJ:SWADMIN): (17)

Cause

Semaphore files may be retained on the system and TIBCO iProcess Objects is unable to create them again as required.

When a semaphore (or a named mutex) is created and used, there are two files on Solaris systems; both file names end with the mutex name (in this case “SWENTOBJ:SWADMIN”) and may also have the node name of the Server as well. One file begins with “.SEML” and the other “.SEMD”.

In one case, if the owner of the TIBCO iProcess Objects Server is changed, the new owner cannot open the mutex.

In the other case, the “.SEML***” file had to be manually removed and then the TIBCO iProcess Objects Server was able to start up with no problem.

These semaphore files “should” be cleared or at least unlocked when the system is re-booted. Since the file names begin with a dot, you need to give the “-a” option to the “ls” command to see them.

Resolution

Remove the “.SEMD” and “.SEML” files in either /var, /var/tmp or /tmp, depending on the operating system involved. Then reboot the UNIX server and restart the TIBCO iProcess Engine and the client application.

“Error in sal_frm_putdata call”

Symptom

This error occurs when starting a case and passing field data.

Cause

The field data you are passing to the TIBCO iProcess Engine is incorrect.

Resolution

Check that the fields you are passing exist and that the data is the correct data type, size, etc.

“One of the items in the array returned an error”

Symptom

This error may occur even if you are attempting to lock a single work item.

Cause

The same error handler is used for the LockItem and LockItems methods.

Resolution

Look in SWWorkItem.LastError for the specific error code.

“The memory could not be ‘written’”

Symptom

When starting the TIBCO iProcess Objects Server, the server seems to start for a few seconds, then an application error occurs with the message “The memory could not be ‘written’”.

Cause

There is a problem within the procedure caching of the TIBCO iProcess Objects Server, where if the addressee of the first step of a procedure is a role, the TIBCO iProcess Objects Server attempts to insert a NULL pointer into the role list and causes an application error.

Resolution

As a workaround, the addressee of the first step should be changed to a normal user, field or group.

“Unable to locate DLL”

Symptom

This may occur near the end of the TIBCO iProcess Objects Server installation. The error message will say that the dynamic link library pthread.dll could not be found in the specified path.

Cause

The installed version of the TIBCO iProcess Engine is not compatible with the version of the TIBCO iProcess Objects Server you are attempting to install.

Resolution

Upgrade the TIBCO iProcess Engine, then reinstall the TIBCO iProcess Objects Server.

“/usr/lib/dld.sl exists - can't open shared library: /oracle8/lib/libclntsh.sl no such file or directory”**Symptom**

This error message appear when starting the TIBCO iProcess Objects Server on a UNIX Oracle variant.

Cause

When using UNIX Oracle variants, the server needs to access shared libraries that are located in the ORACLE_HOME directories. Therefore, the relevant environment variable needs to be set.

Resolution

For AIX, the environment variable \$LIBPATH must be present and must include the "lib" directory of the Oracle server. This is usually \$ORACLE_HOME/lib.

For Solaris, the environment variable \$LD_LIBRARY_PATH must be present and must include the "lib" directory of the Oracle server. This is usually \$ORACLE_HOME/lib.

For HP-UX, the environment variable \$SHLIB_PATH must be present and must include the "lib" directory of the Oracle server. This is usually \$ORACLE_HOME/lib.

“Work Item is not accessible”**Symptom**

This may occur when trying to unlock a work item.

Cause

The work item was locked by the Work Queue Manager (TIBCO iProcess Client), which uses the “lock” mechanism, while the TIBCO iProcess Objects use the "long lock" mechanism. TIBCO iProcess Objects cannot unlock "locked" work items, it can only unlock "long locked" work items.

Resolution

The work item must be unlocked by the Work Queue Manager.

A

Code Examples

Introduction

This appendix provides more comprehensive examples than the “code snippets” you will find in the main body of this document. These examples are provided in the following forms:

- Visual Basic Example
- Java Example
- C++ Example
- Resulting Output

There are links back into the main body of the document where the functionality provided in the examples is described.

The code samples in this appendix are also provided in the form of project files on the distribution CD. This allows you to easily copy any of the code you would like to use in your application. The sample code is located in the **docs/SampleCode** directory on the distribution CD in the following subdirectories:

- **Getting Started** - This includes sample code from the “Getting Started” chapter, including accessing nodes, logging in, logging out, etc.
- **SWLists SWViews SWLocLists** - This includes sample code from the “Working with Lists” chapter. These samples pertain to using SWLists, SWViews, and SWLocLists.
- **SWXLists** - This includes sample code from the “Working with Lists” chapter. These samples pertain to using SWXLists.

Auto-Discovery UDP Broadcast

See [page 22](#) for a description of this functionality.

Visual Basic

Using the “For Each” Iteration:

```
Private Sub RunSample()

    Dim oEnterprise As SWEnterprise
    Dim oNodeInfo As SWNodeInfo

    On Error GoTo Err_RunSample

    Set oEnterprise = New SWEnterprise
    oEnterprise.PollCnt = 2           'Broadcast for 2 sec

    For Each oNodeInfo In oEnterprise.NodeInfos
        ' Display names in intermediate window
        Debug.Print "NodeInfo Key = " & oNodeInfo.Key & vbCrLf _
            & "    IP Address= " & oNodeInfo.IPAddr
    Next
    Exit Sub

Err_RunSample:
    MsgBox "Error Code = " & Err.Number & _
        " Description = " & Err.Description
End Sub
```

Using a While loop:

```
Private Sub cmdRunSample_Click()
    Dim oEnterprise As SWEnterprise
    Dim oNodeInfo As SWNodeInfo
    Dim oList As SWList
    Dim idx As Long

    On Error GoTo Err_RunSample

    Set oEnterprise = New SWEnterprise
    oEnterprise.PollCnt = 2           'Broadcast for 2 sec

    Set oList = oEnterprise.NodeInfos

    idx = 0
    While oList.IsEOL = False
        Set oNodeInfo = oList(idx)
        ' Display names in intermediate window
        Debug.Print "NodeInfo Key = " & oNodeInfo.Key & vbCrLf _
            & "    IP Address= " & oNodeInfo.IPAddr

        idx = idx + 1
    Wend
    Exit Sub

Err_RunSample:
    MsgBox "Error Code = " & Err.Number & _
        " Description = " & Err.Description
End Sub
```

```

Err_RunSample:
    MsgBox "Error Code = " & Err.Number & _
        " Description = " & Err.Description
End Sub

```

Java

Using "Enumeration":

```

import SWEntObj.*;
import java.util.Enumeration;

public class DoSamples
{
    public static void main (String args[]) {

        SWEnterprise oEnterprise;
        SWNodeInfo oNodeInfo;
        Enumeration oListEnum;

        try {
            oEnterprise = new SWEnterprise();
            oEnterprise.setPollCnt((short) 2);
            oListEnum = oEnterprise.getNodeInfos().items();
            for (oListEnum = oEnterprise.getNodeInfos().items();
                oListEnum.hasMoreElements(); ) {
                oNodeInfo = (SWNodeInfo) oListEnum.nextElement();

                System.out.println("NodeInfo Key " + oNodeInfo.getKey() +
                    "\n    IP Address= " + oNodeInfo.getIPAddr());
            }
            System.exit(0); // normal exit
        }

        catch(SWException err) {
            System.out.println("Description = " + err.getMessage());
        }
        catch(Exception err) {
            System.out.println("Description = " + err.getMessage() );
        }
    }
}

```

Using a While Loop:

```

import SWEntObj.*;
import java.util.Enumeration;

public class DoSamples
{
    public static void main (String args[]) {

        SWEnterprise oEnterprise;
        SWNodeInfo oNodeInfo;

```

```

SWList oList;
int idx;

try {
    oEnterprise = new SWEnterprise();
    oEnterprise.setPollCnt((short) 2);
    oList = oEnterprise.getNodeInfos();
    idx = 0;
    while (oList.isEOL() == false) {
        oNodeInfo = (SWNodeInfo) oList.item(idx);

        System.out.println("NodeInfo Key " + oNodeInfo.getKey() +
            "\n      IP Address= " + oNodeInfo.getIPAddr());

        idx = idx + 1;
    }
    System.exit(0); // normal exit
}

catch(SWException err) {
    System.out.println("Description = " + err.getMessage());
}
catch(Exception err) {
    System.out.println("Description = " + err.getMessage() );
}
}
}

```

C++

Note - A "For Next" iteration or enumeration is not available in C++.

Using a While Loop:

```

#include "stdafx.h"
#include <SWEOCPP.h>
#include <ObjTypes.h>
#include <SWObject.h>
#include <SWEnterprise.h>
#include <SWNodeInfo.h>
#include <SWList.h>
#include <SWException.h>

int main(int argc, char* argv[])
{
    SWEnterprise *pEnterprise;
    SWNodeInfo *pNodeInfo;
    SWList *pList;
    long idx;

    try {
        pEnterprise = new SWEnterprise();
        pEnterprise->setPollCnt(2);
        pList = pEnterprise->getNodeInfos();
        idx = 0;

        while(pList->isEOL() == false) {
            pNodeInfo = (SWNodeInfo *) pEnterprise->getNodeInfos()->item(idx);

```

```
        printf("NodeInfo Key %s\n    IP Address= %s\n",
pNodeInfo->getKey(), pNodeInfo->getIPAddr());
        idx++;
    }

}

    catch(SWException err) {
        printf("Error Code = %s Description = %s\n",
err.Number, err.Description);
    }
    catch(...) {
        printf("Unexpected Error\n");
    }

    return 0;
}
```

Resulting Output

```
NodeInfo Key = sw_dana|monti3
    IP Address= 10.20.30.43
NodeInfo Key = rick-srv|swrickf1
    IP Address= 10.20.30.3
NodeInfo Key = bts01|staffw_biz
    IP Address= 10.20.30.4
NodeInfo Key = rick2k|staffw_nod1
    IP Address= 10.20.30.11
NodeInfo Key = listserv|staffware1
    IP Address= 10.20.30.111
```

Directed UDP Broadcast

See [page 25](#) for a description of this functionality.

Visual Basic

```
Private Sub RunSample()

Dim oEnterprise As SWEnterprise

On Error GoTo Err_RunSample

Set oEnterprise = New SWEnterprise
oEnterprise.IsBroadcast = False           'No Broadcast
oEnterprise.AddNode "swdoug2", "doug1" ' add a node

Exit Sub

Err_RunSample:
    MsgBox "Error Code = " & Err.Number & _
        " Description = " & Err.Description
End Sub
```

Java

```
import SWEntObj.*;

public class DoSamples
{

    public static void main (String args[]) {

        SWEnterprise oEnterprise;
        try {
            oEnterprise = new SWEnterprise();
            oEnterprise.setBroadcast(false);
            oEnterprise.addNode("swdoug2", "doug1");           // add a node

        }
        System.exit(0); // normal exit

    }
    catch(SWException err) {
        System.out.println("Description = " + err.getMessage());
    }
    catch(Exception err) {
        System.out.println("Description = " + err.getMessage());
    }
}
}
```

C++

```

#include "stdafx.h"
#include <SWEOCPP.h>
#include <ObjTypes.h>
#include <SWObject.h>
#include <SWEnterprise.h>
#include <SWException.h>

int main(int argc, char* argv[])
{
    SWEnterprise *pEnterprise;

    try {
        pEnterprise = new SWEnterprise();
        pEnterprise->setBroadcast(false);    // No Broadcast
        pEnterprise->addNode("swdoug2", "doug1");    // add a node
    }
    catch(SWException err) {
        printf("Error Code = %s Description = %s\n",
err.Number, err.Description);
    }
    catch(...) {
        printf("Unexpected Error\n");
    }

    return 0;
}

```

Connecting to a Specific Node, Creating Enterprise Users, Login, Logout

This example illustrates all of the following functionality. The list below also provides page numbers on which you can find a description of that particular functionality:

- [Connecting to a Specific Node - page 26](#)
- [Creating Enterprise Users - page 31](#)
- [Logging In - page 32](#)
- [Logging Out - page 34](#)

Visual Basic

```

Private Sub RunSample()
Dim oEnterprise As SWEnterprise
Dim oNodeInfo As SWNodeInfo
Dim oNodeInfoAIX As SWNodeInfo
Dim oEntUser As SWEntUser
Dim oEntUser1 As SWEntUser
Dim oEntUserAdmin As SWEntUser
Dim oNode As SWNode
Dim oNodeSwipe As SWNode

Dim UserArray(2) As String
Dim NodeKeys(1) As String

On Error GoTo Err_RunSample

Set oEnterprise = New SWEnterprise

```

```

oEnterprise.IsBroadcast = False           'no broadcast
Set oNodeInfo = oEnterprise.MakeNodeInfo("seosun2", "swipe", _
                                         "10.20.30.112", 34567)
Set oNodeInfoAIX = oEnterprise.MakeNodeInfo("aixdev", "aixdev", _
                                             "10.20.30.103", 12345) 'add another node
oEnterprise.AddNode "swdoug2", "doug1" ' add a third node

' Create a Enterprise user
Set oEntUserAdmin = oEnterprise.CreateEntUsers("AdminUser")

' Create multiple Enterprise users with a single method call
UserArray(0) = "Edgar"
UserArray(1) = "User1"
UserArray(2) = "Carlos"

oEnterprise.CreateEntUsers UserArray

' print list of SWEntUsers names
Debug.Print "<== List of SWEntUsers ==>"
For Each oEntUser In oEnterprise.EntUsers
    ' Display names in intermediate window
    Debug.Print "SWEntUser Name = " & oEntUser.Name
Next

' Login AdminUser to "swipe" as staffware user "swadmin"
Set oNodeSwipe = oEntUserAdmin.Login(oNodeInfo.Key, "staffware", "swadmin")

' Login AdminUser to 2 more nodes (user is swadmin, password ="staffware")
NodeKeys(0) = oNodeInfoAIX.Key
NodeKeys(1) = "swdoug2|doug1"
oEntUserAdmin.Login NodeKeys, "staffware", "swadmin"

' Login User1 to "swipe" as user "User1" with password "mypassword"
Set oEntUser1 = oEnterprise.EntUsers.ItemByKey("User1")
oEntUser1.Login "seosun2|swipe", "mypassword"

' print list of SWEntUsers and the node key and user name they are logged in as
Debug.Print "<== After logins ... List of SWEntUsers and" & _
           " the nodes they are logged into ==>"
For Each oEntUser In oEnterprise.EntUsers
    ' Display names in intermediate window
    Debug.Print "SWEntUser Name = " & oEntUser.Name & " Logged into " & _
               & oEntUser.LoggedInNodes.Count & " nodes"
    For Each oNode In oEntUser.LoggedInNodes
        Debug.Print " Logged into Node: " & oNode.Key & _
                   " as Staffware User = " & oNode.LoggedInUser.Name
    Next
Next

' logout user1 from swipe node
oEntUser1.Logout           ' calling with no arguments will logout of all ...
                          ' but only logged into a single node in this case

' print list of SWEntUsers and the node key and user name they are logged in as
Debug.Print "<== After User1 logout ... List of SWEntUsers and " & _
           "the nodes they are logged into ==>"
For Each oEntUser In oEnterprise.EntUsers
    ' Display names in intermediate window
    Debug.Print "SWEntUser Name = " & oEntUser.Name & " Logged into " & _

```

```

        oEntUser.LoggedInNodes.Count & " nodes"
    For Each oNode In oEntUser.LoggedInNodes
        Debug.Print " Logged into Node: " & oNode.Key & _
            " as Staffware User = " & oNode.LoggedInUser.Name
    Next
Next
' logout swadmin from swipe node
oEntUserAdmin.Logout "seosun2|swipe"

' print list of SWEntUsers and the node key and user name they are logged in as
Debug.Print "<== After second logout ... List of SWEntUsers and " & _
    & "the nodes they are logged into ==>"
For Each oEntUser In oEnterprise.EntUsers
    ' Display names in intermediate window
    Debug.Print "SWEntUser Name = " & oEntUser.Name & " Logged into " & _
        & oEntUser.LoggedInNodes.Count & " nodes"
    For Each oNode In oEntUser.LoggedInNodes
        Debug.Print " Logged into Node: " & oNode.Key & _
            " as Staffware User = " & oNode.LoggedInUser.Name
    Next
Next
' logout swadmin from all nodes
oEntUserAdmin.Logout

' print list of SWEntUsers and the node key and user name they are logged in as
Debug.Print "<== After all logouts ... List of SWEntUsers and " & _
    "the nodes they are logged into ==>"
For Each oEntUser In oEnterprise.EntUsers
    ' Display names in intermediate window
    Debug.Print "SWEntUser Name = " & oEntUser.Name & " Logged into " & _
        & oEntUser.LoggedInNodes.Count & " nodes"
    For Each oNode In oEntUser.LoggedInNodes
        Debug.Print " Logged into Node: " & oNode.Key & _
            " as Staffware User = " & oNode.LoggedInUser.Name
    Next
Next

Exit Sub

Err_RunSample:
    MsgBox "Error Code = " & Err.Number & _
        " Description = " & Err.Description
End Sub

```

Java

```

import SWEntObj.*;
import java.util.Enumeration;

public class DoSamples
{
    public static void main (String args[]) {

        SWEnterprise oEnterprise;
        SWNodeInfo oNodeInfo;
        SWLocList oLocList;
        SWLocList oLocList1;
        SWNodeInfo oNodeInfoAIX;
        SWEntUser oEntUser;
        SWEntUser oEntUser1;
        SWEntUser oEntUserAdmin;
        SWNode oNode;
        SWNode oNodeSwipe;

        int idx, idx1;
        String[] UserArray;
        String[] NodeKeys;

        try {
            oEnterprise = new SWEnterprise();
            oEnterprise.setBroadcast(false);
            oNodeInfo = oEnterprise.makeNodeInfo("seosun2", "swipe",
                                                "10.20.30.112", 34567);
            oNodeInfoAIX = oEnterprise.makeNodeInfo("aixdev", "aixdev",
                                                  "10.20.30.103", 12345); // add another
            oEnterprise.addNode("swdoug2", "doug1"); // add a third node

            // Create a Enterprise user
            oEntUserAdmin = oEnterprise.createEntUser("AdminUser");

            // Create multiple Enterprise users with a single method call
            UserArray = new String[3];
            UserArray[0] = "Edgar";
            UserArray[1] = "User1";
            UserArray[2] = "Carlos";

            oEnterprise.createEntUsers (UserArray);

            // print list of SWEntUsers names
            System.out.println( "<== List of SWEntUsers ==>");
            oLocList = oEnterprise.getEntUsers();
            idx = 0;
            while (idx < oLocList.count()) {
                oEntUser = (SWEntUser) oLocList.item(idx);
                // Display names in intermediate window
                System.out.println( "SWEntUser Name = " + oEntUser.getName());
                idx++;
            }

            // Login AdminUser to "swipe" as staffware user "swadmin"
            oNodeSwipe = oEntUserAdmin.login(oNodeInfo.getKey(), "staffware", "swadmin");

```

```

// Login AdminUser to 2 more nodes (user is swadmin, password ="staffware")
NodeKeys = new String[2];

NodeKeys[0] = oNodeInfoAIX.getKey();
NodeKeys[1] = "swdoug2|doug1";
oEntUserAdmin.login (NodeKeys, "staffware", "swadmin");

// Login User1 to "swipe" as user "User1" with password "mypassword"
oEntUser1 = (SWEntUser) oEnterprise.getEntUsers().itemByKey("User1");
oEntUser1.login("seosun2|swipe", "mypassword");

// print list of SWEntUsers and node key and user name they are logged in as
System.out.println( "<== After logins ... List of SWEntUsers and " +
    " the nodes they are logged into ==>");

idx = 0;
while (idx < oLocList.count()) {
    oEntUser = (SWEntUser) oLocList.item(idx);
    // Display names in intermediate window
    System.out.println( "SWEntUser Name = " + oEntUser.getName() + " Logged
        into " + oEntUser.getLoggedInNodes().count() + " nodes");
    idx1 = 0;
    oLocList1 = oEntUser.getLoggedInNodes();
    while (idx1 < oLocList1.count()) {
        oNode = (SWNode) oLocList1.item(idx1);
        System.out.println( " Logged into Node: " + oNode.getKey() +
            " as Staffware User = " + oNode.getLoggedInUser().getName());
        idx1++;
    }
    idx++;
}

// logout user1 from swipe node
oEntUser1.logout(); // calling with no arguments will logout of all ...
// but only logged into a single node in this case

// print list of SWEntUsers and node key and user name they are logged in as
System.out.println( "<== After User1 logout ... List of SWEntUsers and " +
    "the nodes they are logged into ==>");

idx = 0;
while (idx < oLocList.count()) {
    oEntUser = (SWEntUser) oLocList.item(idx);
    // Display names
    System.out.println( "SWEntUser Name = " + oEntUser.getName() + " Logged
        into " + oEntUser.getLoggedInNodes().count() + " nodes");
    idx1 = 0;
    oLocList1 = oEntUser.getLoggedInNodes();
    while (idx1 < oLocList1.count()) {
        oNode = (SWNode) oLocList1.item(idx1);
        System.out.println( " Logged into Node: " + oNode.getKey() +
            " as Staffware User = " + oNode.getLoggedInUser().getName());
        idx1++;
    }
    idx++;
}

// logout swadmin from swipe node
oEntUserAdmin.logout("seosun2|swipe");

// print list of SWEntUsers and node key and user name they are logged in as
System.out.println( "<== After second logout ... List of SWEntUsers and "

```

```

        + "the nodes they are logged into ==>");
idx = 0;
while (idx < oLocList.count()) {
    oEntUser = (SWEntUser) oLocList.item(idx);
    // Display names
    System.out.println( "SWEntUser Name = " + oEntUser.getName() + " Logged
        into " + oEntUser.getLoggedInNodes().count() + " nodes");
    idx1 = 0;
    oLocList1 = oEntUser.getLoggedInNodes();
    while (idx1 < oLocList1.count()) {
        oNode = (SWNode) oLocList1.item(idx1);
        System.out.println( " Logged into Node: " + oNode.getKey() +
            " as Staffware User = " + oNode.getLoggedInUser().getName());
        idx1++;
    }
    idx++;
}
// logout swadmin from all nodes
oEntUserAdmin.logout();

// print list of SWEntUsers and node key and user name they are logged in as
System.out.println( "<== After all logouts ... List of SWEntUsers and " +
    "the nodes they are logged into ==>");
idx = 0;
while (idx < oLocList.count()) {
    oEntUser = (SWEntUser) oLocList.item(idx);
    // Display names
    System.out.println( "SWEntUser Name = " + oEntUser.getName() + " Logged
        into " + oEntUser.getLoggedInNodes().count() + " nodes");
    idx1 = 0;
    oLocList1 = oEntUser.getLoggedInNodes();
    while (idx1 < oLocList1.count()) {
        oNode = (SWNode) oLocList1.item(idx1);
        System.out.println( " Logged into Node: " + oNode.getKey() +
            " as Staffware User = " + oNode.getLoggedInUser().getName());
        idx1++;
    }
    idx++;
}

System.exit(0); // normal exit

}
catch(SWException err) {
    System.out.println("Description = " + err.getMessage());
}

catch(Exception err) {
    System.out.println("Description = " + err.getMessage() );
}
}
}

```

C++

```

// UserSamples.cpp : Defines the entry point for the console application.
//

#include "StdAfx.h"
#include <string.h>
#include <SWEOCPP.h>
#include <ObjTypes.h>
#include <SWObject.h>
#include <SWEnterprise.h>
#include <SWEntUser.h>
#include <SWNodeInfo.h>
#include <SWNode.h>
#include <SWUser.h>
#include <SWLocList.h>
#include <SWException.h>

int main(int argc, char* argv[])
{
    SWEnterprise *pEnterprise;
    SWNodeInfo *pNodeInfo;
    SWLocList *pLocList;
    SWLocList *pLocList1;
    SWNodeInfo *pNodeInfoAIX;
    SWEntUser *pEntUser;
    SWEntUser *pEntUser1;
    SWEntUser *pEntUserAdmin;
    SWNode *pNode;
    SWNode *pNodeSwipe;

    int idx, idx1;

    char *UserArray[3];
    char *NodeKeys[2];

    try {
        pEnterprise = new SWEnterprise();
        pEnterprise->setBroadcast(false);
        pNodeInfo = pEnterprise->makeNodeInfo("seosun2", "swipe",
                                             "10.20.30.112", 34567);
        pNodeInfoAIX = pEnterprise->makeNodeInfo("aixdev", "aixdev",
                                                "10.20.30.103", 12345); // add another
        pEnterprise->addNode("swdoug2", "doug1"); // add a third node

        // Create a Enterprise user
        pEntUserAdmin = pEnterprise->createEntUser("AdminUser");

        // Create multiple Enterprise users with a single method call
        UserArray[0] = "Edgar";
        UserArray[1] = "User1";
        UserArray[2] = "Carlos";

        pEnterprise->createEntUsers(3, UserArray);

        // print list pf SWEntUsers names
        printf("<== List of SWEntUsers ==>");
        pLocList = pEnterprise->getEntUsers();
        idx = 0;
        while (idx < pLocList->count()) {

```

```

        pEntUser = (SWEntUser *) pLocList->item(idx);
        // Display names in intermediate window
        printf( "SWEntUser Name = %s", pEntUser->getName());
        idx++;
    }

// Login AdminUser to "swipe" as staffware user "swadmin"
    pNodeSwipe = pEntUserAdmin->login(pNodeInfo->getKey(), "staffware", "swad-
        min");

// Login AdminUser to 2 more nodes (staffware user is swadmin, password ="staff-
        ware")
    NodeKeys[0] = pNodeInfoAIX->getKey();
    NodeKeys[1] = "swdoug2|doug1";
    pEntUserAdmin->login(2, NodeKeys, "staffware", "swadmin");

// Login User1 to "swipe" as staffware user "User1" with password "mypassword"
    pEntUser1 = (SWEntUser *) pEnterprise->getEntUsers()->itemByKey("User1");
    pEntUser1->login("seosun2|swipe", "mypassword");

// print list of SWEntUsers and the node key and user name they are logged in as
    printf("<== After logins ... List of SWEntUsers and"
        " the nodes they are logged into ==>");
    idx = 0;
    while (idx < pLocList->count()) {
        pEntUser = (SWEntUser *) pLocList->item(idx);
        // Display names in intermediate window
        printf( "SWEntUser Name = %s Logged into %d nodes", pEntUser->getName(),
            pEntUser->getLoggedInNodes()->count());

        idx1 = 0;
        pLocList1 = pEntUser->getLoggedInNodes();
        while (idx1 < pLocList1->count()) {
            pNode = (SWNode *) pLocList1->item(idx1);
            printf(" Logged into Node: %s as Staffware User = %s", pNode->getKey(),
                pNode->getLoggedInUser()->getName());
            idx1++;
        }
        idx++;
    }

// logout user1 from swipe node
    pEntUser1->logout(); // calling with no arguments will logout of all
        // but only logged into a single node in this case

// print list of SWEntUsers and the node key and user name they are logged in as
    printf( "<== After User1 logout ... List of SWEntUsers and "
        "the nodes they are logged into ==>");
    idx = 0;
    while (idx < pLocList->count()) {
        pEntUser = (SWEntUser *) pLocList->item(idx);
        // Display names
        printf( "SWEntUser Name = %s Logged into %d nodes", pEntUser->getName(),
            pEntUser->getLoggedInNodes()->count());

        idx1 = 0;
        pLocList1 = pEntUser->getLoggedInNodes();
        while (idx1 < pLocList1->count()) {
            pNode = (SWNode *) pLocList1->item(idx1);
            printf(" Logged into Node: %s as Staffware User = %s", pNode->getKey(),
                pNode->getLoggedInUser()->getName());
            idx1++;
        }
    }

```

```

    }
    idx++;
}

// logout swadmin from swipe node
pEntUserAdmin->logout("seosun2|swipe");

// print list of SWEntUsers and node key and user name they are logged in as
printf( "<== After second logout ... List of SWEntUsers and "
        "the nodes they are logged into ==>");
idx = 0;
while (idx < pLocList->count()) {
    pEntUser = (SWEntUser *) pLocList->item(idx);
    // Display names
    printf( "SWEntUser Name = %s Logged into %d nodes", pEntUser->getName(),
            pEntUser->getLoggedInNodes()->count());
    idx1 = 0;
    pLocList1 = pEntUser->getLoggedInNodes();
    while (idx1 < pLocList1->count()) {
        pNode = (SWNode *) pLocList1->item(idx1);
        printf( " Logged into Node: %s as Staffware User = %s", pNode->getKey(),
                pNode->getLoggedInUser()->getName());
        idx1++;
    }
    idx++;
}

// logout swadmin from all nodes
pEntUserAdmin->logout();

// print list of SWEntUsers and node key and user name they are logged in as
printf("<== After all logouts ... List of SWEntUsers and "
        "the nodes they are logged into ==>");
idx = 0;
while (idx < pLocList->count()) {
    pEntUser = (SWEntUser *) pLocList->item(idx);
    // Display names
    printf( "SWEntUser Name = %s Logged into %d nodes", pEntUser->getName(),
            pEntUser->getLoggedInNodes()->count());
    idx1 = 0;
    pLocList1 = pEntUser->getLoggedInNodes();
    while (idx1 < pLocList1->count()) {
        pNode = (SWNode *) pLocList1->item(idx1);
        printf(" Logged into Node: %s as Staffware User = %s", pNode->getKey(),
                pNode->getLoggedInUser()->getName());
        idx1++;
    }
    idx++;
}
}

catch(SWException err) {
printf("Error Code = %d Description = %s\n", err.Number, err.Description);
}
catch(...) {
printf("Unexpected Error\n");
}

return 0;
}

```

Resulting Output

```
<== List of SWEntUsers ==>
SWEntUser Name = AdminUser
SWEntUser Name = Edgar
SWEntUser Name = User1
SWEntUser Name = Carlos
<== After logins ... List of SWEntUsers and the nodes they are logged into ==>
SWEntUser Name = AdminUser Logged into 3 nodes
  Logged into Node: seosun2|swipe as Staffware User = swadmin
  Logged into Node: aixdev|aixdev as Staffware User = swadmin
  Logged into Node: SWDOUG2|doug1 as Staffware User = swadmin
SWEntUser Name = Edgar Logged into 0 nodes
SWEntUser Name = User1 Logged into 1 nodes
  Logged into Node: seosun2|swipe as Staffware User = User1
SWEntUser Name = Carlos Logged into 0 nodes
<== After User1 logout ... List of SWEntUsers and the nodes they are logged into ==>
SWEntUser Name = AdminUser Logged into 3 nodes
  Logged into Node: seosun2|swipe as Staffware User = swadmin
  Logged into Node: aixdev|aixdev as Staffware User = swadmin
  Logged into Node: SWDOUG2|doug1 as Staffware User = swadmin
SWEntUser Name = Edgar Logged into 0 nodes
SWEntUser Name = User1 Logged into 0 nodes
SWEntUser Name = Carlos Logged into 0 nodes
<== After second logout ... List of SWEntUsers and the nodes they are logged into ==>
SWEntUser Name = AdminUser Logged into 2 nodes
  Logged into Node: aixdev|aixdev as Staffware User = swadmin
  Logged into Node: SWDOUG2|doug1 as Staffware User = swadmin
SWEntUser Name = Edgar Logged into 0 nodes
SWEntUser Name = User1 Logged into 0 nodes
SWEntUser Name = Carlos Logged into 0 nodes
<== After all logouts ... List of SWEntUsers and the nodes they are logged into ==>
SWEntUser Name = AdminUser Logged into 0 nodes
SWEntUser Name = Edgar Logged into 0 nodes
SWEntUser Name = User1 Logged into 0 nodes
SWEntUser Name = Carlos Logged into 0 nodes
```

Working with Staffware Lists — SWLists, SWViews, and SWLocLists

The examples in this section illustrate the following functionality. The list below provides page numbers on which you can find a description of functionality illustrated in the examples.

- Forcing synchronous behavior - see [page 57](#)
- Determining the number of objects in a list - [page 58](#)
- Creating objects and adding them to an SWList - [page 61](#)
- Adding objects/strings to SWLocLists - [page 63](#)
- Accessing items on SWLocLists - [page 64](#)
- Creating objects and adding them to an SWView - [page 66](#)
- Rebuilding an SWView - [page 69](#)
- Specifying filter criteria in an SWView - [page 154](#)
- Specifying sort criteria on an SWView - [page 180](#)

Visual Basic

```
Option Explicit
' To run this sample will need to change the hard-coded servername, nodename, and user
' Also need to change the name of the work queue and the work queue you use should have
' at least 100 workitems on it

Private Sub cmdRunSample_Click()
Dim oEnterprise As SWEnterprise
Dim oNodeInfo As SWNodeInfo
Dim oEntUser As SWEntUser
Dim oNode As SWNode
Dim oWorkQ As SWWorkQ
Dim oWorkQList As SWList
Dim oUser As SWUser
Dim oWorkItems As SWView
Dim oWorkItem As SWWorkItem
Dim oSortField As SWSortField
Dim oField As SWField

Dim i As Integer
Dim cnt As Integer
Dim tag As String
Dim key As String

On Error GoTo Err_RunSample

    Debug.Print "<@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ SAMPLE START @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@>"
    Set oEnterprise = New SWEnterprise
    oEnterprise.IsBroadcast = False           'no broadcast
    Set oNodeInfo = oEnterprise.MakeNodeInfo("swdoug2", "doug1", _
                                             "10.20.30.108", 3908)

    ' Create a Enterprise user to represent swadmin
    Set oEntUser = oEnterprise.CreateEntUsers("swadmin")

    ' Login swadmin into node
    Set oNode = oEntUser.Login(oNodeInfo.key, "staffware")

    ' All work will be done based on scope (i.e. permissions) of logged on user
    Set oUser = oNode.LoggedInUser
```

```

' Display WorkQ names and description in intermediate window
' Using IsWaitForAll (i.e. synchronous messaging)
' Print list of WorkQs to which swadmin belongs
  Debug.Print "<=== Iterating list of WorkQs with IsWaitForAll set true ===>"
  Set oWorkQList = oUser.WorkQs
  oWorkQList.IsWaitForAll = True
  For Each oWorkQ In oWorkQList
      ' Since first access, msg sent to server on this
      ' statement will not return from this
      ' statement on first access until all message
      ' buffers have been returned (NOTE: ' this statement
      ' does check for isEOL under the covers
      Debug.Print "Name: " & oWorkQ.Name & "; Description: " _
        & oWorkQ.Description
  Debug.Print "oWorkQList.Count=" & oWorkQList.Count ' Current number SWProc
' Objects in list
  Next
  ' Total number of WorkQs
  Debug.Print "After iterating List oWorkQList.Count = " & oWorkQList.Count

' Display WorkQ names and description in intermediate window
' Not Using IsWaitForAll (i.e. asynchronous messaging)
' Print list of WorkQs to which swadmin belongs
  Debug.Print "<=== Re-Iterating List of WorkQs with IsWaitForAll set false ===>"
  oWorkQList.Rebuild
      ' WorkQ list is cleared locally and new msg sent
      ' to server

  oWorkQList.IsWaitForAll = False
  i = 0
      ' alternate syntax (equivalent to "For each" above)
  oWorkQList.IsEOL = False
      ' Not necessary since Rebuild resets this value but
      ' but if comment out rebuild then would be needed to ensure
      ' that we would re-iterate through the list
  While (oWorkQList.IsEOL = False)
      ' Since message sent by rebuild, no message
      ' sent on this statement
      ' If replace oWorkQList.Rebuild with a oWorkQList.Clear
      ' then msg to server would have been sent on this statement
      ' 1st item is parced here to see if IsEOL is true
      Debug.Print "Name: " & oWorkQList(i).Name & "; Description: " _
        & oWorkQList(i).Description
      ' same as oWorkQList.item(i).Description
      ' gets next item out of msg buffer, does not wait until
      ' all buffers from server are received
      Debug.Print "oWorkQList.Count = " & oWorkQList.Count
      ' Current number SWProc
      ' Objects in list

      i = i + 1
  Wend

  ' Total number of WorkQs
  Debug.Print "After iterating list oWorkQList.Count = " & oWorkQList.Count

'Using ItemByKey
'For larger lists, it is much quicker to use the Stateless object calls
' (such as MakeWorkQ method on Node instead of doing an ItemByKey)
Debug.Print "<=== Accessing specific WorkQ ===>"
Set oWorkQ = oWorkQList.ItemByKey("swadmin@doug1|R")
Debug.Print "Queue (from ItemByKey) description : " & oWorkQ.Description
Set oWorkQ = oNode.MakeWorkQ("swadmin", True, "doug1")
Debug.Print "Queue (from MakeWorkQ) description : " & oWorkQ.Description

```

```

'Using RebuildAll
Debug.Print "<=== Rebuilding WorkQ with RebuildALL ===>"
oWorkQ.IsRebuildAll = True
oWorkQ.Rebuild      ' Will cause messages to be sent to the server
                   ' to update ALL lists on oWorkQ (i.e Rebuild called)

' The 2 previous lines are equivalent to the following 8 lines of code
oWorkQ.IsRebuildAll = False
oWorkQ.Rebuild      ' Will cause messages to be sent to the server
oWorkQ.Participations.Rebuild
oWorkQ.SupervisorNames.Rebuild
oWorkQ.ViewUserNames.Rebuild
oWorkQ.WorkItems.Rebuild
oWorkQ.WorkItemsX.Rebuild
oWorkQ.CaseDataQParamDefs.Rebuild

'Adding strings & objects to a Local List
' Also demonstrates setting sort and filter criteria for view
Debug.Print "<=== Adding strings & objects to a Local List ===>"
Set oWorkItems = oWorkQ.WorkItems
oWorkItems.Clear
oWorkItems.MaxCnt = 20
Set oWorkItem = oWorkItems(0)      ' msg sent since first access after clear
Debug.Print "Field Count (for workitem:" & oWorkItem.key & ") = "_
           & oWorkItem.Case.Fields.Count      ' No fields returned for Case associated
           ' with Workitem
oWorkItems.CaseFieldNames.Clear      ' ensure empty local list before adding
oWorkItems.CaseFieldNames.Add "SW_CASENUM" 'return Staffware Case Number Field
oWorkItems.CaseFieldNames.Add "TESTPROFILED3"      ' return User defined Field

' Enable workitems to be sorted by Casenum
For Each oSortField In oWorkItems.SortFields ' Note there are sortfields
           ' present by default
    Debug.Print "Sortfield Field Name = " & oSortField.FieldName
Next

'clear current list since want to sort ONLY by casenum
oWorkItems.SortFields.Clear

' Create and configure Sortfield
Set oSortField = New SWSortField
oSortField.FieldName = "SW_CASENUM"
oSortField.IsAscending = False
oSortField.SortAsType = swNumericSort

' Add SortField to local list
oWorkItems.SortFields.Add oSortField

Debug.Print "<=== After adding fieldnames and sort criteria on View===>"
i = 0
oWorkItems.IsEOL = False

```

```

' Accessing workitems in view
While Not (oWorkItems.IsEOL) ' Setting View iteration
  Set oWorkItem = oWorkItems(i)
  Debug.Print "WorkItem Key= " & oWorkItem.key & ", CaseNum = "_
    & oWorkItem.Case.CaseNumber & ", Fields returned = "_
    & oWorkItem.Case.Fields.Count
  For Each oField In oWorkItem.Case.Fields
    Debug.Print "    FieldName = " & oField.Name & ", Field Value = "_
    & oField.Value
  Next
  i = i + 1
Wend
oWorkItems.Rebuild ' Causes new message to be sent to server
                  ' so criteria takes effect
Debug.Print "<=== After Rebuild with fieldnames and sort criteria ===>"
For Each oWorkItem In oWorkItems
  Debug.Print "WorkItem Key= " & oWorkItem.key & ", CaseNum = "_
    & oWorkItem.Case.CaseNumber & ", Fields returned = "_
    & oWorkItem.Case.Fields.Count
  For Each oField In oWorkItem.Case.Fields
    Debug.Print "    FieldName = " & oField.Name & ", Field Value = "_
    & oField.Value
  Next
Next

' Get only workitems of procedure TestPro4
oWorkItems.FilterExpression = "SW_PRONAME = ""TestPro4""
oWorkItems.Rebuild
Debug.Print "<=== After View Rebuild with fieldnames and sort criteria ===>"
For Each oWorkItem In oWorkItems
  Debug.Print "WorkItem Key= " & oWorkItem.key & ", CaseNum = "_
    & oWorkItem.Case.CaseNumber & ", Fields returned = "_
    & oWorkItem.Case.Fields.Count
  For Each oField In oWorkItem.Case.Fields
    Debug.Print "    FieldName = " & oField.Name & ", Field Value = "_
    & oField.Value
  Next
Next

tag = oWorkItems(2).tag ' Save tag from workitem for ItemByKey example below
key = oWorkItems(2).key ' Save key from workitem for ItemByKey example below

'Accessing a Local List
Debug.Print "<=== Accessing a Local List ===>"
cnt = oWorkItems.CaseFieldNames.Count 'No msg sent to server so can loop on count
For i = 0 To cnt - 1
  Debug.Print "CaseField Name = " & oWorkItems.CaseFieldNames(i)
Next

Set oSortField = oWorkItems.SortFields.ItemByKey("SW_CASENUM")
Debug.Print "SortField Field Name = " & oSortField.FieldName

'Accessing item in View with ItemByKey
Debug.Print "Show properties of workitem returned with ItemByKey"
Set oWorkItem = oWorkItems.ItemByKey(key)
Debug.Print "Tag = " & oWorkItem.tag
Debug.Print "MailId = " & oWorkItem.MailID
Debug.Print "Case Number = " & oWorkItem.Case.CaseNumber
Debug.Print "Procedure Name = " & oWorkItem.Case.ProcName
Debug.Print "Case Description = " & oWorkItem.Case.Description

```

```

    Debug.Print "Show properties of workitem returned using MakeWorkItemByTag"
    Set oWorkItem = oNode.MakeWorkItemByTag(tag) ' returns same workitem as one
                                                ' returned with ItemByKey above

    Debug.Print "Tag = " & oWorkItem.tag
    Debug.Print "MailId = " & oWorkItem.MailID
    Debug.Print "Case Number = " & oWorkItem.Case.CaseNumber
    Debug.Print "Procedure Name = " & oWorkItem.Case.ProcName
    Debug.Print "Case Description = " & oWorkItem.Case.Description

' Disconnect swadmin TCP/IP session from node
oEntUser.Disconnect ' ends TCP/IP connection and removes oNode
                    ' from oEnterprise.EntUser.LoggedInNodes

Exit Sub

Err_RunSample:
    MsgBox "Error Code = " & Err.Number & _
          " Description = " & Err.Description
End Sub

```

Java

```

import SWEntObj.*;
import java.util.Enumeration;

public class DoSamples
{
// To run this sample will need to change the hard-coded servername, nodename, and
// user. Also need to change the name of the work queue and the work queue you use
// should have at least 100 workitems on it

    public static void main (String args[]) {

        SWEnterprise oEnterprise;
        SWNodeInfo oNodeInfo;
        SWEntUser oEntUser;
        SWNode oNode;
        SWWorkQ oWorkQ;
        SWList oWorkQList;
        SWUser oUser;
        SWView oWorkItems;
        SWWorkItem oWorkItem;
        SWSortField oSortField;
        SWField oField;
        Enumeration oListEnum;
        Enumeration oListEnum1;

        int i, cnt;
        String tag, key;

        try {
            System.out.println( "<@@@@@@@@@@@@@@@@@ SAMPLE START @@@@@@@@@@@@@@@@@@>");
            oEnterprise = new SWEnterprise();
            oEnterprise.setBroadcast(false);

```

```

oNodeInfo = oEnterprise.makeNodeInfo("swdoug2", "doug1",
                                     "10.20.30.108", 4498);

// Create a Enterprise user to represent swadmin
oEntUser = oEnterprise.createEntUser("swadmin");

// Login swadmin
oNode = oEntUser.login(oNodeInfo.getKey(), "staffware");
// All work will be done based on scope (i.e. permissions) of logged on user
oUser = oNode.getLoggedInUser();

// Display WorkQ names and description in intermediate window
// Using IsWaitForAll (i.e. synchronous messaging)
// Print list of WorkQs to which swadmin belongs
System.out.println("<=== Iterating list of WorkQs with IsWaitForAll set
                    true ==>");
oWorkQList = oUser.getWorkQs();
oWorkQList.setWaitForAll(true);
for (oListEnum = oWorkQList.items(); oListEnum.hasMoreElements(); ) {
    // Since first access, msg sent to server on this statement
    oWorkQ = (SWWorkQ) oListEnum.nextElement();
        // Will not return from this statement on first access until
        // all message buffers have been returned (NOTE: this statement
        // does check for isEOL under the covers
    System.out.println("Name: " + oWorkQ.getName() + "; Description: "
                      + oWorkQ.getDescription());
    System.out.println("oWorkQList.getCount() = " + oWorkQList.count());
    // Current number SWProc Objects in list
}

System.out.println("After iterating List oWorkQList.Count = " +
                  oWorkQList.count());    // Total number of WorkQs

// Display WorkQ names and description in intermediate window
// Not Using IsWaitForAll (i.e. asynchronous messaging)
// Print list of WorkQs to which swadmin belongs
System.out.println("<=== Re-Iterating List of WorkQs with IsWaitForAll
                    set false ==>");
oWorkQList.rebuild();    // WorkQ list is cleared locally and new msg sent
                        // to server
oWorkQList.setWaitForAll(false);
i = 0;    // alternate syntax (equivalent to "For each" above)
oWorkQList.setEOL(false);    // Not necessary since Rebuild resets this
                            // value but if comment out rebuild then would be
                            // needed to ensure that we would re-iterate
                            // through the list
while (oWorkQList.isEOL() == false) {    // Since message sent by rebuild,
    // no message sent on this statement
    // If replace oWorkQList.Rebuild with a
    // oWorkQList.Clear then msg to
    // server would have been sent on this statement
    // 1st item is parsed here to see if IsEOL is true
    oWorkQ = (SWWorkQ) oWorkQList.item(i);
    System.out.println("Name: " + oWorkQ.getName() + "; Description: "
                      + oWorkQ.getDescription()); // same as oWorkQList.item(i).Description
                                                // gets next item out of msg buffer, does
                                                // not wait until all buffers from server a
                                                //re received

```

```

        System.out.println("oWorkQList.Count = " + oWorkQList.count());
        // Current number SWProc Objects in list
        i = i + 1;
    }

    System.out.println("After iterating list oWorkQList.Count = " +
        oWorkQList.count()); // Total number of WorkQs

    // Using ItemByKey
    // For larger lists, it is much quicker to use the Stateless object calls
    // (such as MakeWorkQ method on Node instead of doing an ItemByKey)
    System.out.println("<=== Accessing specific WorkQ ===>");
    oWorkQ = (SWWorkQ) oWorkQList.itemByKey("swadmin@doug1|R");
    System.out.println("Queue (from ItemByKey) description : " +
        oWorkQ.getDescription());
    oWorkQ = oNode.makeWorkQ("swadmin", true, "doug1");
    System.out.println("Queue (from MakeWorkQ) description : " +
        oWorkQ.getDescription());

    // Using RebuildAll
    System.out.println("<=== Rebuilding WorkQ with RebuildALL ===>");
    oWorkQ.setRebuildAll(true);
    oWorkQ.rebuild(); // Will cause messages to be sent to the server
                    // to update ALL lists on oWorkQ (i.e Rebuild called)

    // The 2 previous lines are equivalent to the following 8 lines of code
    oWorkQ.setRebuildAll(false);
    oWorkQ.rebuild(); // Will cause messages to be sent to the server
    oWorkQ.getParticipations().rebuild();
    oWorkQ.getSupervisorNames().rebuild();
    oWorkQ.getViewUserNames().rebuild();
    oWorkQ.getWorkItems().rebuild();
    oWorkQ.getWorkItemsX().rebuild();
    oWorkQ.getCaseDataQParamDefs().rebuild();

    // Adding strings & objects to a Local List
    // Also demonstrates setting sort and filter criteria for view
    System.out.println("<=== Adding strings + objects to a Local List ===>");
    oWorkItems = oWorkQ.getWorkItems();
    oWorkItems.clear();
    oWorkItems.setMaxCnt(20);
    oWorkItem = (SWWorkItem) oWorkItems.item(0); // msg sent since first
                                                // access after clear
    System.out.println("Field Count (for workitem:" + oWorkItem.getKey() + ") = "
        + oWorkItem.getCase().getFields().count());
    oWorkItems.getCaseFieldNames().clear(); // ensure empty local list
                                           // before adding
    oWorkItems.getCaseFieldNames().add("SW_CASEENUM"); // return Staffware Case
                                                       // Number Field
    oWorkItems.getCaseFieldNames().add("TESTPROFILED3"); // return User defined
                                                         // Field

    // Enable workitems to be sorted by Casenumber
    for (oListEnum = oWorkItems.getSortFields().items();
        oListEnum.hasMoreElements(); ) {
        // Note there are sortfields present by default
        oSortField = (SWSortField) oListEnum.nextElement();
        System.out.println("Sortfield Field Name = " + oSortField.getFieldName());
    }

```

```

//clear current list since want to sort ONLY by casenum
oWorkItems.getSortFields().clear();

// Create and configure Sortfield
oSortField = new SWSortField();
oSortField.setFieldName("SW_CASENUM");
oSortField.setAscending(false);
oSortField.setSortAsType(SWSortType.swNumericSort);
// Add SortField to local list
oWorkItems.getSortFields().add( oSortField);

System.out.println("<=== After adding fieldnames and sort criteria
                    on View===>");

i = 0;
oWorkItems.setEOL(false);

// Accessing workitems in view
while (oWorkItems.isEOL() == false) { // Setting View iteration
    oWorkItem = (SWWorkItem) oWorkItems.item(i);
    System.out.println("WorkItem Key= " + oWorkItem.getKey()
        + ", CaseNum = " + oWorkItem.getCASE().getCaseNumber()
        + ", Fields returned = "
        + oWorkItem.getCASE().getFields().count());
    for (oListEnum1 = oWorkItem.getCASE().getFields().items();
        oListEnum1.hasMoreElements(); ) {
        oField = (SWField) oListEnum1.nextElement();
        System.out.println("    FieldName = " + oField.getName()
            + ", Field Value = " + oField.getValue());
    }
    i = i + 1;
}

oWorkItems.rebuild(); // Causes new message to be sent to server so
                    // criteria takes effect
System.out.println("<=== After Rebuild with fieldnames and sort
                    criteria ===>");
for (oListEnum = oWorkItems.items(); oListEnum.hasMoreElements(); ) {
    oWorkItem = (SWWorkItem) oListEnum.nextElement();
    System.out.println("WorkItem Key= " + oWorkItem.getKey()
        + ", CaseNum = " + oWorkItem.getCASE().getCaseNumber()
        + ", Fields returned = "
        + oWorkItem.getCASE().getFields().count());
    for (oListEnum1 = oWorkItem.getCASE().getFields().items();
        oListEnum1.hasMoreElements(); ) {
        oField = (SWField) oListEnum1.nextElement();
        System.out.println("    FieldName = " + oField.getName()
            + ", Field Value = " + oField.getValue());
    }
}

// Get only workitems of procedure TestPro4
oWorkItems.setFilterExpression("SW_PRONAME = \"TestPro4\"");
oWorkItems.rebuild();
System.out.println("<=== After View Rebuild with fieldnames and sort
                    criteria ===>");
for (oListEnum = oWorkItems.items(); oListEnum.hasMoreElements(); ) {
    oWorkItem = (SWWorkItem) oListEnum.nextElement();

```

```

        System.out.println("WorkItem Key= " + oWorkItem.getKey()
            + ", CaseNum = "
            + oWorkItem.getCase().getCaseNumber()
            + ", Fields returned = " +
            oWorkItem.getCase().getFields().count());
        for (oListEnum1 = oWorkItem.getCase().getFields().items();
            oListEnum1.hasMoreElements(); ) {
            oField = (SWField) oListEnum1.nextElement();
            System.out.println("    FieldName = " + oField.getName()
                + ", Field Value = " + oField.getValue());
        }
    }

oWorkItem = (SWWorkItem) oWorkItems.item(2);

tag = oWorkItem.getTag(); //Save tag from workitem for ItemByKey example below
key = oWorkItem.getKey(); //Save key from workitem for ItemByKey example below

// Accessing a Local List
System.out.println("<=== Accessing a Local List ===>");
cnt = oWorkItems.getCaseFieldNames().count(); //No msg sent to server so
// can loop on count
for ( i = 0; i < cnt - 1; i++) {
    System.out.println("CaseField Name = "
        + oWorkItems.getCaseFieldNames().item(i));
}

oSortField = (SWSortField) oWorkItems.getSortFields().itemByKey("SW_CASENUM");
System.out.println("SortField Field Name = " + oSortField.getFieldName());

// Accessing item in View with ItemByKey
System.out.println("Show properties of workitem returned with ItemByKey");
oWorkItem = (SWWorkItem) oWorkItems.itemByKey(key);
System.out.println("Tag = " + oWorkItem.getTag());
System.out.println("MailId = " + oWorkItem.getMailID());
System.out.println("Case Number = " + oWorkItem.getCase().getCaseNumber());
System.out.println("Procedure Name = " + oWorkItem.getCase().getProcName());
System.out.println("Case Description = " +
    oWorkItem.getCase().getDescription());

System.out.println("Show properties of workitem returned using
    MakeWorkItemByTag");
oWorkItem = oNode.makeWorkItemByTag(tag); // returns same workitem as one
// returned with ItemByKey above
System.out.println("Tag = " + oWorkItem.getTag());
System.out.println("MailId = " + oWorkItem.getMailID());
System.out.println("Case Number = " + oWorkItem.getCase().getCaseNumber());
System.out.println("Procedure Name = " + oWorkItem.getCase().getProcName());
System.out.println("Case Description = " +
    oWorkItem.getCase().getDescription());

// disconnect swadmin from node
oEntUser.disconnect();

System.exit(0); // normal exit

```

```

    }
    catch(SWException err) {
        System.out.println("Description = " + err.getMessage());
    }

    catch(Exception err) {
        System.out.println("Description = " + err.getMessage() );
    }
}
}

```

C++

```

#include "StdAfx.h"

#include <time.h>
#include <string.h>

#include <SWEOCPP.h>
#include <ObjTypes.h>
#include <SWObject.h>
#include <SWEnterprise.h>
#include <SWEntUser.h>
#include <SWNodeInfo.h>
#include <SWNode.h>
#include <SWUser.h>
#include <SWLocList.h>
#include <SWList.h>
#include <SWView.h>
#include <SWXList.h>
#include <SWWorkQ.h>
#include <SWCase.h>
#include <SWSortField.h>
#include <SWField.h>
#include <SWWorkItem.h>
#include <SWException.h>

int main(int argc, char* argv[])
{
    SWEnterprise *pEnterprise;
    SWNodeInfo *pNodeInfo;
    SWEntUser *pEntUser;
    SWNode *pNode;
    SWWorkQ *pWorkQ;
    SWList *pWorkQList;
    SWUser *pUser;
    SWView *pWorkItems;
    SWWorkItem *pWorkItem;
    SWSortField *pSortField;
    SWField *pField;
    SWFieldType fldType;

    int i, j, cnt;
    char *pTag = NULL;
    char *pKey = NULL;
    char txtField[256];
    double NumValue;

    try {
        printf( "<@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ SAMPLE START @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@>\n");
        pEnterprise = new SWEnterprise();
        pEnterprise->setBroadcast(false);
        pNodeInfo = pEnterprise->makeNodeInfo("swdoug2", "doug1",
            "10.20.30.108", 3908);
    }
}

```

```

// Create a Enterprise user o represent swadmin
    pEntUser = pEnterprise->createEntUser("swadmin");

// Login swadmin
    pNode = pEntUser->login(pNodeInfo->getKey(), "staffware");
// All work will be done based on scope (i.e. permissions) of logged on user
    pUser = pNode->getLoggedInUser();

// Display WorkQ names and description in intermediate window
// Using IsWaitForAll (i.e. synchronous messaging)
// Print list of WorkQs to which swadmin belongs
printf("<=== Iterating list of WorkQs with IsWaitForAll set true ===>\n");
pWorkQList = pUser->getWorkQs();
pWorkQList->setWaitForAll(true);
i = 0;
while (pWorkQList->isEOL() == false) {
    // Since first access, msg sent to server on this statement
    pWorkQ = (SWWorkQ *) pWorkQList->item(i);
        // Will not return from this statement on first access until
        // all message buffers have been returned (NOTE: this statement
        // does check for isEOL under the covers
    printf("Name: %s; Description: %s\n", pWorkQ->getName(), pWorkQ->getDescription());
    printf("pWorkQList->getCount() = %d\n", pWorkQList->count()); // Current number
        // SWProc Objects in list
    i++;
}

printf("After iterating List pWorkQList->Count = %d\n", pWorkQList->count());
    // Total number of WorkQs

// Display WorkQ names and description in intermediate window
// Not Using IsWaitForAll (i.e. asynchronous messaging)
// Print list of WorkQs to which swadmin belongs
printf("<=== Re-Iterating List of WorkQs with IsWaitForAll set false ===>\n");
pWorkQList->rebuild(); // WorkQ list is cleared locally and new msg sent to server
pWorkQList->setWaitForAll(false);
i = 0; // alternate syntax (equivalent to "For each" above)
pWorkQList->setEOL(false); // Not necessary since Rebuild resets this value but
    // but if comment out rebuild then would be needed to ensure
    // that we would re-iterate through the list
while (pWorkQList->isEOL() == false) { // Since message sent by rebuild,
    // no message sent on this statement
    // If replace oWorkQList.Rebuild with a oWorkQList->Clear then msg to
    // server would have been sent on this statement
    // 1st item is parced here to see if IsEOL is true
    pWorkQ = (SWWorkQ *) pWorkQList->item(i);
    printf("Name: %s; Description: %d\n", pWorkQ->getName(),
        pWorkQ->getDescription());
        // same as oWorkQList->item(i)->Description
        // gets next item out of msg buffer, does not wait until
        // all buffers from server are received
    printf("pWorkQList->Count = %d\n", pWorkQList->count());
        // Current number SWProc Objects in list
    i++;
}

printf("After iterating list pWorkQList->Count = %d\n", pWorkQList->count());
    // Total number of WorkQs

```

```

//Using ItemByKey
//For larger lists, it is much quicker to use the Stateless object calls
// (such as MakeWorkQ method on Node instead of doing an ItemByKey)
printf("<=== Accessing specific WorkQ ===>\n");
pWorkQ = (SWorkQ *) pWorkQList->itemByKey("swadmin@doug1|R");
    printf("Queue (from ItemByKey) description : %s\n", pWorkQ->getDescription());

    SWorkQ *pWorkQ1 = new SWorkQ();
pWorkQ1 = pNode->makeWorkQ(pWorkQ1, "swadmin", true, "doug1");
    printf("Queue (from MakeWorkQ) description : %s\n", pWorkQ1->getDescription());
    delete pWorkQ1;

//Using RebuildAll
printf("<=== Rebuilding WorkQ with RebuildALL ===>\n");
pWorkQ->setRebuildAll(true);
pWorkQ->rebuild();          // Will cause messages to be sent to the server
                          // to update ALL lists on oWorkQ (i.e Rebuild called)
// The 2 previous lines are equivalent to the following 8 lines of code
pWorkQ->setRebuildAll(false);
    pWorkQ->rebuild();          // Will cause messages to be sent to the server
pWorkQ->getParticipations()->rebuild();
pWorkQ->getSupervisorNames()->rebuild();
pWorkQ->getViewUserNames()->rebuild();
pWorkQ->getWorkItems()->rebuild();
pWorkQ->getWorkItemsX()->rebuild();
pWorkQ->getCaseDataQParamDefs()->rebuild();

//Adding strings & objects to a Local List
// Also demonstrates setting sort and filter criteria for view
printf("<=== Adding strings + objects to a Local List ===>\n");
pWorkItems = pWorkQ->getWorkItems();
pWorkItems->clear();
pWorkItems->setMaxCnt(20);
    pWorkItem = (SWorkItem *) pWorkItems->item(0);    // msg sent since first access
                                                // after clear
printf("Field Count (for workitem: %s) = %d\n",pWorkItem->getKey(),
        pWorkItem->getCase()->getFields()->count());
    pWorkItems->getCaseFieldNames()->clear();    // ensure empty local list before adding
pWorkItems->getCaseFieldNames()->add( "SW_CASENUM");    // return Staffware Case
                                                // Number Field
    pWorkItems->getCaseFieldNames()->add("TESTPROFILED3");    // return User defined Field
// Enable workitems to be sorted by Casenum
i = 0;
cnt = pWorkItems->getSortFields()->count();
while (i < cnt) {
    // Note there are sortfields present by default
    pSortField = (SSortField *) pWorkItems->getSortFields()->item(i);
    printf("Sortfield Field Name = %s\n",pSortField->getFieldName());
    i++;
}
//clear current list since want to sort ONLY by casenum
pWorkItems->getSortFields()->clear();

// Create and configure Sortfield
pSortField = new SSortField( "SW_CASENUM", false, swNumericSort);
// Add SortField to local list
pWorkItems->getSortFields()->add( pSortField); // DO NOT delete since NOT copied

printf("<=== After adding fieldnames and sort criteria on View===>\n");
i = 0;
pWorkItems->setEOL(false);

```

```

// Accessing workitems in view
while (pWorkItems->isEOL() == false) { //Setting View iteration
    pWorkItem = (SWorkItem *) pWorkItems->item(i);
    printf("WorkItem Key= %s, CaseNum = %d, Fields returned = %d\n",
        pWorkItem->getKey(),
        pWorkItem->getCase()->getCaseNumber(),
        pWorkItem->getCase()->getFields()->count());
    j = 0;
    while (pWorkItem->getCase()->getFields()->isEOL() == false) {
        pField = (SWField *) pWorkItem->getCase()->getFields()->item(j);
        FldType =pField->getType();
        switch(FldType) {
            case swNumericAttr:
                pField->getValue(NumValue);
                printf("    FieldName = %s, Field Value = %d\n", pField->getName(),
                    NumValue);
                break;

            case swTextAttr:
                pField->getValue(txtField, sizeof(txtField));
                printf("    FieldName = %s, Field Value = %s\n", pField->getName(),
                    txtField);
                break;

            default:
                printf("<==== Unexpect Field type ..sample procs had ONLY numeric or
                    text fields =====>\n");
        }
        j++;
    }
    i++;
}
pWorkItems->rebuild(); // Causes new message to be sent to server so
//criteria takes effect
printf("<==== After Rebuild with fieldnames and sort criteria ====>\n");
i = 0;
while (pWorkItems->isEOL() == false ) {
    pWorkItem = (SWorkItem *) pWorkItems->item(i);
    printf("WorkItem Key= %s, CaseNum = %d, Fields returned = %d\n",pWorkItem->getKey(),
        pWorkItem->getCase()->getCaseNumber(),
        pWorkItem->getCase()->getFields()->count());
    j = 0;
    while (pWorkItem->getCase()->getFields()->isEOL() == false) {
        pField = (SWField *) pWorkItem->getCase()->getFields()->item(j);
        FldType =pField->getType();
        switch(FldType) {
            case swNumericAttr:
                pField->getValue(NumValue);
                printf("    FieldName = %s, Field Value = %d\n", pField->getName(),
                    NumValue);
                break;

            case swTextAttr:
                pField->getValue(txtField, sizeof(txtField));
                printf("    FieldName = %s, Field Value = %s\n", pField->getName(),
                    txtField);
                break;

            default:
                printf("<==== Unexpect Field type ..sample procs had ONLY numeric
                    or text fields =====>\n");
        }
        j++;
    }
    i++;
}
}

```

```

// Get only workitems of procedure TestPro4
pWorkItems->setFilterExpression("SW_PRONAME = \"TestPro4\"");
pWorkItems->rebuild();
printf("<=== After View Rebuild with fieldnames and sort criteria ===>\n");
i = 0;
while (pWorkItems->isEOL() == false) {
    pWorkItem = (SWWorkItem *) pWorkItems->item(i);
    printf("WorkItem Key= %s, CaseNum = %d, Fields returned = %d\n",
        pWorkItem->getKey(),
        pWorkItem->getCase()->getCaseNumber(),
        pWorkItem->getCase()->getFields()->count());
    j = 0;
    while (pWorkItem->getCase()->getFields()->isEOL() == false) {
        pField = (SWField *) pWorkItem->getCase()->getFields()->item(j);
        FldType = pField->getType();
        switch(FldType) {
            case swNumericAttr:
                pField->getValue(NumValue);
                printf("    FieldName = %s, Field Value = %d\n",
                    pField->getName(), NumValue);
                break;

            case swTextAttr:
                pField->getValue(txtField, sizeof(txtField));
                printf("    FieldName = %s, Field Value = %s\n",
                    pField->getName(), txtField);
                break;

            default:
                printf("<==== Unexpect Field type ..sample procs had ONLY
                    numeric or text fields ====>\n");
        }
        j++;
    }
    i++;
}

pWorkItem = (SWWorkItem *) pWorkItems->item(2);
pTag = pWorkItem->getTag(); // Save tag from workitem for accessing item example below
pKey = pWorkItem->getKey(); // Save tag from workitem for accessing item example below
// Accessing a Local List
printf("<=== Accessing a Local List ===>\n");
cnt = pWorkItems->getCaseFieldNames()->count(); //No msg sent to server so can
//loop on count
for ( i = 0; i < cnt - 1; i++) {
    printf("CaseField Name = %s\n", pWorkItems->getCaseFieldNames()->item(i));
}

pSortField = (SWSortField *) pWorkItems->getSortFields()->itemByKey("SW_CASENUM");
printf("SortField Field Name = %s\n", pSortField->getFieldName());

//Accessing item in View with ItemByKey
printf("Show properties of workitem returned with ItemByKey\n");
pWorkItem = (SWWorkItem *) pWorkItems->itemByKey(pKey);
printf("Tag = %s\n", pWorkItem->getTag());
printf("MailId = %s\n", pWorkItem->getMailID());
printf("Case Number = %d\n", pWorkItem->getCase()->getCaseNumber());
printf("Procedure Name = %s\n", pWorkItem->getCase()->getProcName());
printf("Case Description = %s\n", pWorkItem->getCase()->getDescription());

printf("Show properties of workitem returned using MakeWorkItemByTag\n");
SWWorkItem *pWorkItem1 = new SWWorkItem();
pWorkItem1 = pNode->makeWorkItemByTag(pWorkItem1, pTag); // returns same workitem as
//one returned with ItemByKey above

```

```

printf("Tag = %s\n", pWorkItem1->getTag());
printf("MailId = %s\n", pWorkItem1->getMailID());
printf("Case Number = %d\n", pWorkItem1->getCase()->getCaseNumber());
printf("Procedure Name = %s\n", pWorkItem1->getCase()->getProcName());
printf("Case Description = %s\n", pWorkItem1->getCase()->getDescription());
delete pWorkItem1;

// disconnect swadmin from node
pEntUser->disconnect(); // calling with no arguments will logout of all ...
                        // but only logged into a single node in this case
                        //delete pEnterprise;

}

catch(SWException err) {
    printf("Error Code = %n Description = %s\n", err.Number, err.Description);
}
catch(...) {
    printf("Unexpected Error\n");
}

return 0;
}

```

Resulting Output

```

VB Test Output
<@@@@@@@@@@@@@@@@@@@@@ SAMPLE START @@@@@@@@@@@@@@@@@@@@@>
<=== Iterating list of WorkQs with IsWaitForAll set true ===>
Name: STUDENTS; Description: STUDENTS
oWorkQList.Count = 2
Name: swadmin; Description: System Administrator
oWorkQList.Count = 3
Name: TESTGROUP1; Description: TESTGROUP1
oWorkQList.Count = 4
Name: TESTGROUP2; Description: TESTGROUP2
oWorkQList.Count = 5
Name: TESTGROUP3; Description: TESTGROUP3
oWorkQList.Count = 6
Name: TESTGROUP4; Description: TESTGROUP4
oWorkQList.Count = 7
Name: TESTGROUP5; Description: TESTGROUP5
oWorkQList.Count = 8
Name: TESTGROUP6; Description: TESTGROUP6
oWorkQList.Count = 8
After iterating List oWorkQList.Count = 8
<=== Re-Iterating List of WorkQs with IsWaitForAll set false ===>
Name: STUDENTS; Description: STUDENTS
oWorkQList.Count = 2
Name: swadmin; Description: System Administrator
oWorkQList.Count = 3
Name: TESTGROUP1; Description: TESTGROUP1
oWorkQList.Count = 4
Name: TESTGROUP2; Description: TESTGROUP2
oWorkQList.Count = 5
Name: TESTGROUP3; Description: TESTGROUP3
oWorkQList.Count = 6
Name: TESTGROUP4; Description: TESTGROUP4
oWorkQList.Count = 7
Name: TESTGROUP5; Description: TESTGROUP5
oWorkQList.Count = 8
Name: TESTGROUP6; Description: TESTGROUP6
oWorkQList.Count = 8
After iterating list oWorkQList.Count = 8

```

```

<=== Accessing specific WorkQ ===>
Queue (from ItemByKey) description : System Administrator
Queue (from MakeWorkQ) description : System Administrator
<=== Rebuilding WorkQ with RebuildALL ===>
<=== Adding strings & objects to a Local List ===>
Field Count (for workitem:swadmin@dougl|9420803) = 0
Sortfield Field Name = SW_HOSTNAME
Sortfield Field Name = SW_PRONAME
Sortfield Field Name = SW_CASENUM
Sortfield Field Name = SW_STEPNAME
<=== After adding fieldnames and sort criteria on View===>
WorkItem Key= swadmin@dougl|9420803, CaseNum = 1, Fields returned = 0
WorkItem Key= swadmin@dougl|9416707, CaseNum = 2, Fields returned = 0
WorkItem Key= swadmin@dougl|9412611, CaseNum = 3, Fields returned = 0
WorkItem Key= swadmin@dougl|9408515, CaseNum = 4, Fields returned = 0
WorkItem Key= swadmin@dougl|9404419, CaseNum = 5, Fields returned = 0
WorkItem Key= swadmin@dougl|9400323, CaseNum = 6, Fields returned = 0
WorkItem Key= swadmin@dougl|9396227, CaseNum = 7, Fields returned = 0
WorkItem Key= swadmin@dougl|9392131, CaseNum = 8, Fields returned = 0
WorkItem Key= swadmin@dougl|9388035, CaseNum = 9, Fields returned = 0
WorkItem Key= swadmin@dougl|9383939, CaseNum = 10, Fields returned = 0
WorkItem Key= swadmin@dougl|9379843, CaseNum = 11, Fields returned = 0
WorkItem Key= swadmin@dougl|9375747, CaseNum = 12, Fields returned = 0
WorkItem Key= swadmin@dougl|9371651, CaseNum = 13, Fields returned = 0
WorkItem Key= swadmin@dougl|9367555, CaseNum = 14, Fields returned = 0
WorkItem Key= swadmin@dougl|9363459, CaseNum = 15, Fields returned = 0
WorkItem Key= swadmin@dougl|9359363, CaseNum = 16, Fields returned = 0
WorkItem Key= swadmin@dougl|9355267, CaseNum = 17, Fields returned = 0
WorkItem Key= swadmin@dougl|9351171, CaseNum = 18, Fields returned = 0
WorkItem Key= swadmin@dougl|9347075, CaseNum = 19, Fields returned = 0
WorkItem Key= swadmin@dougl|9342979, CaseNum = 20, Fields returned = 0
<=== After Rebuild with fieldnames and sort criteria ===>
WorkItem Key= swadmin@dougl|5234690, CaseNum = 1034, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5238786, CaseNum = 1033, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5242882, CaseNum = 1032, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5246978, CaseNum = 1031, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5251074, CaseNum = 1030, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5255170, CaseNum = 1029, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5259266, CaseNum = 1028, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5263362, CaseNum = 1027, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5267458, CaseNum = 1026, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5271554, CaseNum = 1025, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5275650, CaseNum = 1024, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5279746, CaseNum = 1023, Fields returned = 2

```

```

    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5283842, CaseNum = 1022, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5287938, CaseNum = 1021, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5292034, CaseNum = 1020, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5296130, CaseNum = 1019, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5300226, CaseNum = 1018, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5304322, CaseNum = 1017, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5308418, CaseNum = 1016, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5312514, CaseNum = 1015, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
<=== After View Rebuild with fieldnames and sort criteria ===>
WorkItem Key= swadmin@dougl|7012356, CaseNum = 6, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value = Rubbish
WorkItem Key= swadmin@dougl|7016452, CaseNum = 5, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value = Primo
WorkItem Key= swadmin@dougl|7020547, CaseNum = 4, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value = fair
WorkItem Key= swadmin@dougl|7024646, CaseNum = 3, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value = poor
WorkItem Key= swadmin@dougl|7028741, CaseNum = 2, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value = excellent
<=== Accessing a Local List ===>
CaseField Name = SW_CASENUM
CaseField Name = TESTPROFIELD3
SortField Field Name = SW_CASENUM
Show properties of workitem returned with ItemByKey
Tag = dougl|TESTPRO4|swadmin|R|4|7020547
MailId = 7020547
Case Number = 4
Procedure Name = TESTPRO4
Case Description = Third Test Case
Show properties of workitem returned using MakeWorkItemByTag
Tag = dougl|TESTPRO4|swadmin|R|4|7020547
MailId = 7020547
Case Number = 4
Procedure Name = TESTPRO4
Case Description = Third Test Case

```

Working with Staffware Lists — SWXLists

The examples in this section illustrate the following functionality. The list below provides page numbers on which you can find a description of functionality illustrated in the examples.

- Populating an SWXList of work items - [page 77](#)
- Determining the number of items in an SWXList - [page 80](#)
- Persisting an SWXList - [page 81](#)
- Specifying sort criteria for an SWXList - [page 180](#)

Visual Basic

```
Option Explicit
' To run this sample will need to change the hard-coded servername, nodename, and user
' Also need to change the name of the work queue and the work queue you use should
' have at least 100 workitems on it

Private Sub cmdRunSample_Click()
Dim oEnterprise As SWEnterprise
Dim oNodeInfo As SWNodeInfo
Dim oEntUser As SWEntUser
Dim oNode As SWNode
Dim oWorkQ As SWWorkQ
Dim oWorkQList As SWList
Dim oUser As SWUser
Dim oWorkItemsX As SWXList
Dim oWorkItem As SWWorkItem
Dim oSortField As SWSortField
Dim oField As SWField

Dim i As Integer
Dim cnt As Integer
Dim tag As String
Dim key As String
Dim XListId As String
Dim blksize As Integer

Dim Fieldnames() As String
Dim SortFlds As Variant
Dim SortFldArray() As SWSortField

Dim UserArray(2) As String
Dim NodeKeys(1) As String

On Error GoTo Err_RunSample

Debug.Print "<@@@@@@@@@@@@@@@@ SAMPLE XLIST START @@@@@@@@@@@@@@@@@@>"
Set oEnterprise = New SWEnterprise
oEnterprise.IsBroadcast = False           'no broadcast
Set oNodeInfo = oEnterprise.MakeNodeInfo("swdoug2", "doug1", _
                                         "10.20.30.108", 3908)

' Create a Enterprise user to swadmin
Set oEntUser = oEnterprise.CreateEntUsers("swadmin")

' Login swadmin to node
Set oNode = oEntUser.Login(oNodeInfo.key, "staffware")
```

```

Set oWorkQ = oNode.MakeWorkQ("swadmin", True, "doug1")

' Adding strings & objects to XList Criteria
Debug.Print "<=== Adding strings & objects to a XList Criteria ===>"
Set oWorkItemsX = oWorkQ.WorkItemsX
  Set oWorkItem = oWorkItemsX(0)          ' msg sent since first access after clear
' No fields returned for Case associated with Workitem
  Debug.Print ("Field Count (for workitem:" & oWorkItem.key & ") = "_
    & oWorkItem.Case.Fields.Count)
ReDim Fieldnames(1)
Fieldnames(0) = "SW_CASENUM"              ' return Staffware Case Number Field
Fieldnames(1) = "TESTPROFIELD3"          ' return User defined Field
oWorkItemsX.Criteria.CaseFieldNames = Fieldnames ' replaces existing array of
    ' fieldnames
' Enable WorkItemsX to be sorted by Casenumber

Debug.Print "<=== Show list of preset/default sortfields (ie see std client) ===>"
cnt = UBound(oWorkItemsX.Criteria.SortFields)
For i = LBound(oWorkItemsX.Criteria.SortFields) To cnt
  SortFlds = oWorkItemsX.Criteria.SortFields
  Set oSortField = SortFlds(i)
  Debug.Print ("Sortfield Field Name = " & oSortField.FieldName)
Next

' Create and configure Sortfield
Set oSortField = New SWSortField
oSortField.FieldName = "SW_CASENUM"
oSortField.IsAscending = False
oSortField.SortAsType = swNumericSort
' Add SortField
ReDim SortFldArray(0)
Set SortFldArray(0) = oSortField
oWorkItemsX.Criteria.SortFields = SortFldArray
Debug.Print "<=== List sortfields after configuring criteria on Xlist ===>"
cnt = UBound(oWorkItemsX.Criteria.SortFields)
For i = LBound(oWorkItemsX.Criteria.SortFields) To cnt
  SortFlds = oWorkItemsX.Criteria.SortFields
  Set oSortField = SortFlds(i)
  Debug.Print ("Sortfield Field Name = " & oSortField.FieldName)
Next

Debug.Print "<=== First 25 workitems after adding setting Xlist Criteria ===>"
Debug.Print "Total number of workitems on xlist on server = "_
  & oWorkItemsX.ItemCount
i = 0

' Accessing WorkItemsX in XList
' Get only 1st 25 back
If oWorkItemsX.ItemCount > 25 Then
  cnt = 25 ' if more than 25 only loop through 1st 25
Else
  cnt = oWorkItemsX.ItemCount ' if less than 25 loop through all of them
End If
While (i < cnt)
  Set oWorkItem = oWorkItemsX(i)
  Debug.Print ("WorkItem Key= " & oWorkItem.key & ", CaseNum = "_
    & oWorkItem.Case.CaseNumber & ", Fields returned = "_
    & oWorkItem.Case.Fields.Count)

```

```

    For Each oField In oWorkItem.Case.Fields
        Debug.Print ("      fieldName = " & oField.Name & ", Field Value = "_
            & oField.Value)
    Next
    i = i + 1
Wend
oWorkItemsX.Clear                ' Workaround for CR10994
blksize = 5
oWorkItemsX.Rebuild blksize      ' Causes new message to be sent to server
                                ' so criteria takes effect
Debug.Print "<=== First 25 workitems after Rebuild with new block size_
            (blk size = 5)===>"
Debug.Print "Total number of workitems on xlist on server = "_
            & oWorkItemsX.ItemCount
cnt = 0
For Each oWorkItem In oWorkItemsX
    Debug.Print ("WorkItem Key= " & oWorkItem.key & ", CaseNum = "_
        & oWorkItem.Case.CaseNumber & ", Fields returned = "_
        & oWorkItem.Case.Fields.Count)
    For Each oField In oWorkItem.Case.Fields
        Debug.Print ("      fieldName = " & oField.Name & ", Field Value = "_
            & oField.Value)
    Next
    cnt = cnt + 1
    If cnt > 25 Then
        Exit For                ' only display first 25 or less
    End If
    If (cnt Mod blksize) = 0 Then ' minimize memory use on client
        Debug.Print "      XList clear "
        oWorkItemsX.Clear      ' the block of 5 since already displayed info
    End If
Next
Debug.Print "<=== Local workitem count after listing first 25 workitems _
            with clears ===>"
Debug.Print " Local number of workitems = " & oWorkItemsX.Count

' Get only WorkItemsX of procedure TestPro4
oWorkItemsX.Criteria.FilterExpression = "SW_PRONAME = ""TestPro4""
oWorkItemsX.Rebuild
Debug.Print "<=== First 25 workitems after XList Rebuild with filter_
            expression ===>"
cnt = 0
For Each oWorkItem In oWorkItemsX
    Debug.Print ("WorkItem Key= " & oWorkItem.key & ", CaseNum = "_
        & oWorkItem.Case.CaseNumber & ", Fields returned = "_
        & oWorkItem.Case.Fields.Count)
    For Each oField In oWorkItem.Case.Fields
        Debug.Print ("      fieldName = " & oField.Name & ", Field Value = "_
            & oField.Value)
    Next
    cnt = cnt + 1
    If cnt > 25 Then
        Exit For                ' only display first 25 or less
    End If
Next
tag = oWorkItemsX(2).tag        ' Save tag from workitem for ItemByKey example below
key = oWorkItemsX(2).key        ' Save key from workitem for ItemByKey example below

```

```

'Accessing item in SWXList with ItemByKey
Debug.Print "Show properties of workitem returned with ItemByKey"
Set oWorkItem = oWorkItemsX.ItemByKey(key)
Debug.Print "Tag = " & oWorkItem.tag
Debug.Print "MailId = " & oWorkItem.MailID
Debug.Print "Case Number = " & oWorkItem.Case.CaseNumber
Debug.Print "Procedure Name = " & oWorkItem.Case.ProcName
Debug.Print "Case Description = " & oWorkItem.Case.Description

Debug.Print "Show properties of workitem returned using MakeWorkItemByTag"
Set oWorkItem = oNode.MakeWorkItemByTag(tag) ' returns same workitem as one
' returned with ItemByKey above

Debug.Print "Tag = " & oWorkItem.tag
Debug.Print "MailId = " & oWorkItem.MailID
Debug.Print "Case Number = " & oWorkItem.Case.CaseNumber
Debug.Print "Procedure Name = " & oWorkItem.Case.ProcName
Debug.Print "Case Description = " & oWorkItem.Case.Description

' Create a XList to persist for future use
Set oWorkItemsX = oWorkQ.MakeXListItems(10) ' will return 10 items per block
Debug.Print "Number of Items in Xlist= " & oWorkItemsX.ItemCount
Debug.Print "Display the tag of first 5 workitems in XList"
i = 0
While i < 5
    Set oWorkItem = oWorkItemsX(i)
    Debug.Print " WorkItem key = " & oWorkItem.tag
    i = i + 1
Wend
Debug.Print " Number of items on Xlist after getting first 5" & oWorkItemsX.Count
Debug.Print "Display the tag of last 10 workitems in XList"
i = oWorkItemsX.ItemCount - 10
While i < oWorkItemsX.ItemCount
    Set oWorkItem = oWorkItemsX(i)
    Debug.Print " WorkItem key = " & oWorkItem.tag
    i = i + 1
Wend
Debug.Print "After getting last 10 workitems on XList (blk size = 5)"
Debug.Print "Number of items on local Xlist = " & oWorkItemsX.Count

' Persist XList
oWorkItemsX.Criteria.IsPersisted = True          msg sent to server to keep list
XListId = oWorkItemsX.Criteria.PersistenceId
Debug.Print "SWXList Persisted Id = " & XListId
Set oWorkItemsX = Nothing                        ' release XList

' Disconnect swadmin TCP/IP session from node
oEntUser.Disconnect          ' ends TCP/IP connection and removes oNode
                            ' from oEnterprise.EntUser.LoggedInNodes

' Re-connect (ie logon)
Set oNode = oEntUser.Login(oNodeInfo.key, "staffware")

' Re-attach to persisted XList
Set oWorkItemsX = oNode.GetXList(XListId)
Debug.Print "Retrieve persisted XList: item count = " & oWorkItemsX.ItemCount

```

```

Debug.Print "Before getting last 10 workitems on persisted XList_
            (def blk size = 20)"
Debug.Print "Display the workitem tags"
i = oWorkItemsX.ItemCount - 10
While i < oWorkItemsX.ItemCount
    Set oWorkItem = oWorkItemsX(i)
    Debug.Print " WorkItem key = " & oWorkItem.tag
    i = i + 1
Wend
Debug.Print "After getting last 10 workitems on persisted XList_
            (def blk size = 20)"
Debug.Print "Number of items on local Xlist = " & oWorkItemsX.Count

Exit Sub

Err_RunSample:
    MsgBox "Error Code = " & Err.Number & _
        " Description = " & Err.Description
End Sub

```

Java

```

import SWEntObj.*;
import java.util.Enumeration;

public class DoSamples
{
    // To run this sample you will need to change the hard-coded servername, nodename, and
    // user. Also need to change the name of the work queue and the work queue you use
    // should have at least 100 workitems on it

    public static void main (String args[]) {

        SWEnterprise oEnterprise;
        SWNodeInfo oNodeInfo;
        SWEntUser oEntUser;
        SWNode oNode;
        SWWorkQ oWorkQ;
        SWList oWorkQList;
        SWUser oUser;
        SWXList oWorkItemsX;
        SWWorkItem oWorkItem;
        SWSortField oSortField;
        SWField oField;
        SWCriteriaWI oCriteriaWI;
        Enumeration oListEnum;

        int i, cnt, blksize;
        String tag, key, XListId;
        String[] FieldNames;
        SWSortField[] SortFldArray;

        try {
            System.out.println( "<@@@@@@@@@@@@ SAMPLE XLIST START @@@@@@@@@@@@@@@@@@@@@>");
            oEnterprise = new SWEnterprise();
            oEnterprise.setBroadcast(false);

```

```

        oNodeInfo = oEnterprise.makeNodeInfo("swdoug2", "doug1",
                                           "10.20.30.108", 4498);

// Create a Enterprise user to represent swadmin
    oEntUser = oEnterprise.createEntUser("swadmin");

// Login swadmin to node
    oNode = oEntUser.login(oNodeInfo.getKey(), "staffware");

    oWorkQ = oNode.makeWorkQ("swadmin", true, "doug1");

// Adding strings & objects to XList Criteria
    System.out.println("<=== Adding strings + objects to a XList Criteria ===>");
    oWorkItemsX = oWorkQ.getWorkItemsX();
    oWorkItem = (SWWorkItem) oWorkItemsX.item(0);    // msg sent since first
                                                    // access after clear
// No fields returned for Case associated with Workitem
    System.out.println("Field Count (for workitem:" + oWorkItem.getKey() + ") = "
        + oWorkItem.getCase().getFields().count());
    FieldNames = new String[2];
    FieldNames[0] = "SW_CASENUM";    // return Staffware Case Number Field
    FieldNames[1] = "TESTPROFIELD3"; // return User defined Field
    oCriteriaWI = (SWCriteriaWI) oWorkItemsX.getCriteria();
    oCriteriaWI.setCaseFieldNames(FieldNames);    //replaces existing array of
                                                    //fieldnames

// Enable WorkItemsX to be sorted by Casenumber

System.out.println("<=== Show list of preset/default sortfields
                    (ie see std client) ===>");
SortFldArray = oCriteriaWI.getSortFields();
cnt = SortFldArray.length;
i = 0;
while( i < cnt) { // Note there are sortfields present by default
    oSortField = SortFldArray[i];
    System.out.println("Sortfield Field Name = " + oSortField.getFieldName());
    i++;
}
// Create and configure Sortfield
oSortField = new SWSortField();
oSortField.setFieldName("SW_CASENUM");
oSortField.setAscending(false);
oSortField.setSortAsType(SWSortType.swNumericSort);
// Add SortField
SortFldArray = new SWSortField[1];
SortFldArray[0] = oSortField;
oCriteriaWI.setSortFields(SortFldArray);
System.out.println("<=== List sortfields after configuring criteria
                    on Xlist ===>");
cnt = SortFldArray.length;
i = 0;
while (i < cnt) {
    oSortField = SortFldArray[i];
    System.out.println("Sortfield Field Name = " + oSortField.getFieldName());
    i++;
}
System.out.println("<=== First 25 workitems after adding setting Xlist
                    Criteria ===>");
System.out.println("Total number of workitems on xlist on server = "
    + oWorkItemsX.getItemCount());

```

```

// Accessing WorkItemsX in XList
// Get only 1st 25 back
if (oWorkItemsX.getItemCount() > 25) {
    cnt = 25; // if more than 25 only loop through 1st 25
}
else {
    cnt = oWorkItemsX.getItemCount(); //if less than 25 loop through all of them
}
i = 0;
while (i < cnt) {
    oWorkItem = (SWWorkItem) oWorkItemsX.item(i);
    System.out.println("WorkItem Key= " + oWorkItem.getKey() + ", CaseNum = " +
        oWorkItem.getCase().getCaseNumber() + ", Fields
        returned = " + oWorkItem.getCase().getFields().count());
    for (oListEnum = oWorkItem.getCase().getFields().items();
        oListEnum.hasMoreElements(); ) {
        oField = (SWField) oListEnum.nextElement();
        System.out.println("    FieldName = " + oField.getName() + ",
            Field Value = " + oField.getValue());
    }
    i = i + 1;
}
oWorkItemsX.clear(); // Workaround for CR10994
blksize = 5;
oWorkItemsX.rebuild( blksize); // Causes new message to be sent to
// server so criteria takes effect
System.out.println("<=== First 25 workitems after Rebuild with new block
    size (blk size = 5)===>");
System.out.println("Total number of workitems on xlist on server = " +
    oWorkItemsX.getItemCount());
i = 0;
if (oWorkItemsX.getItemCount() > 25) {
    cnt = 25; // if more than 25 only loop through 1st 25
}
else {
    cnt = oWorkItemsX.getItemCount(); //if less than 25 loop through all of them
}
while (i < cnt) {
    oWorkItem = (SWWorkItem) oWorkItemsX.item(i);
    System.out.println("WorkItem Key= " + oWorkItem.getKey() + ", CaseNum = " +
        oWorkItem.getCase().getCaseNumber() +
        ", Fields returned = " + oWorkItem.getCase().getFields().count());
    for (oListEnum = oWorkItem.getCase().getFields().items();
        oListEnum.hasMoreElements(); ) {
        oField = (SWField) oListEnum.nextElement();
        System.out.println("    FieldName = " + oField.getName() + ",
            Field Value = " + oField.getValue());
    }
    i++;
    if (cnt%blksize == 0) { // minimize memory use on client
        System.out.println("    XList clear ");
        oWorkItemsX.clear(); // the block of 5 since already displayed info
    }
}

System.out.println("<=== Local workitem count after listing first 25
    workitems with clears ===>");
System.out.println(" Local number of workitems = " + oWorkItemsX.count());
// Get only WorkItemsX of procedure TestPro4
oCriteriaWI = (SWCriteriaWI) oWorkItemsX.getCriteria();

```

```

oCriteriaWI.setFilterExpression( "SW_PRONAME = \"TestPro4\"");
oWorkItemsX.rebuild();
System.out.println("<== First 25 workitems after XList Rebuild with
                    filter expression ==>");

i = 0;
if (oWorkItemsX.getItemCount() > 25) {
    cnt = 25;           // if more than 25 only loop through 1st 25
}
else {
    cnt = oWorkItemsX.getItemCount(); // if less than 25 loop through all of them
}
while (i < cnt) {
    oWorkItem = (SWWorkItem) oWorkItemsX.item(i);
    System.out.println("WorkItem Key= " + oWorkItem.getKey() + ", CaseNum = "
        + oWorkItem.getCase().getCaseNumber()+ ", Fields returned = " +
        oWorkItem.getCase().getFields().count());
    for (oListEnum = oWorkItem.getCase().getFields().items();
        oListEnum.hasMoreElements(); ) {
        oField = (SWField) oListEnum.nextElement();
        System.out.println("    FieldName = " + oField.getName() + ",
            Field Value = " + oField.getValue());
    }
    i++;
}
oWorkItem = (SWWorkItem) oWorkItemsX.item(2);
tag = oWorkItem.getTag(); // Save tag from workitem for ItemByKey example below
key = oWorkItem.getKey(); // Save key from workitem for ItemByKey example below

//Accessing item in SWXList with ItemByKey
System.out.println("Show properties of workitem returned with ItemByKey");
oWorkItem = (SWWorkItem) oWorkItemsX.itemByKey(key);
System.out.println("Tag = " + oWorkItem.getTag());
System.out.println("MailId = " + oWorkItem.getMailID());
System.out.println("Case Number = " + oWorkItem.getCase().getCaseNumber());
System.out.println("Procedure Name = " + oWorkItem.getCase().getProcName());
System.out.println("Case Description = " + oWorkItem.getCase().getDescription());

System.out.println("Show properties of workitem returned using
                    MakeWorkItemByTag");
oWorkItem = oNode.makeWorkItemByTag(tag); // returns same workitem as one
// returned with ItemByKey above
System.out.println("Tag = " + oWorkItem.getTag());
System.out.println("MailId = " + oWorkItem.getMailID());
System.out.println("Case Number = " + oWorkItem.getCase().getCaseNumber());
System.out.println("Procedure Name = " + oWorkItem.getCase().getProcName());
System.out.println("Case Description = " + oWorkItem.getCase().getDescription());

// Create a XList to persist for future use
oWorkItemsX = oWorkQ.makeXListItems(10); // will return 10 items per block
System.out.println("Number of Items in Xlist= " + oWorkItemsX.getItemCount());
System.out.println("Display the tag of first 5 workitems in XList");
i = 0;
if (oWorkItemsX.getItemCount() > 5)
    cnt = 5;
else
    cnt = oWorkItemsX.getItemCount();
while (i < cnt) {

```

```

        oWorkItem = (SWWorkItem) oWorkItemsX.item(i);
        System.out.println(" WorkItem tag = " + oWorkItem.getTag());
        i++;
    }

    System.out.println(" Number of items on Xlist after getting first 5" +
        oWorkItemsX.count());
    System.out.println("Display the tag of last 10 workitems in XList");
    i = oWorkItemsX.getItemCount() - 10;
    while (i < oWorkItemsX.getItemCount()){
        oWorkItem = (SWWorkItem) oWorkItemsX.item(i);
        System.out.println(" WorkItem key = " + oWorkItem.getTag());
        i++;
    }

    System.out.println("After getting last 10 workitems on XList (blk size = 5)");
    System.out.println("Number of items on local Xlist = " + oWorkItemsX.count());

    // Persist XList
    oCriteriaWI = (SWCriteriaWI) oWorkItemsX.getCriteria();
    oCriteriaWI.setIsPersisted(true); // msg sent to server to keep list
    XListId = oCriteriaWI.getPersistenceId();
    System.out.println("SWXList Persisted Id = " + XListId);

    // Disconnect swadmin TCP/IP session from node
    oEntUser.disconnect(); // ends TCP/IP connection and removes oNode
    // from oEnterprise.EntUser.LoggedInNodes

    // Re-connect (ie logon)
    oNode = oEntUser.login(oNodeInfo.getKey(), "staffware");

    // Re-attach to persisted XList
    oWorkItemsX = oNode.getXList(XListId);
    System.out.println("Retrieve persisted XList: item count = " +
        oWorkItemsX.getItemCount());

    System.out.println("Before getting last 10 workitems on persisted XList (def
        blk size = 20)");
    System.out.println("Display the workitem tags");
    i = oWorkItemsX.getItemCount() - 10;
    while (i < oWorkItemsX.getItemCount()) {
        oWorkItem = (SWWorkItem) oWorkItemsX.item(i);
        System.out.println(" WorkItem key = " + oWorkItem.getTag());
        i++;
    }

    System.out.println("After getting last 10 workitems on persisted XList (def
        blk size = 20)");
    System.out.println("Number of items on local Xlist = " + oWorkItemsX.count());

    System.exit(0); // normal exit

    }
    catch(SWException err) {
        System.out.println("Description = " + err.getMessage());
    }
}

```

```

        catch(Exception err) {
            System.out.println("Description = " + err.getMessage() );
        }
    }
}

```

C++

```

#include "StdAfx.h"

#include <time.h>
#include <string.h>

#include <SWEOCPP.h>
#include <ObjTypes.h>
#include <SWObject.h>
#include <SWEnterprise.h>
#include <SWEntUser.h>
#include <SWNodeInfo.h>
#include <SWNode.h>
#include <SWUser.h>
#include <SWLocList.h>
#include <SWList.h>
#include <SWView.h>
#include <SWXList.h>
#include <SWWorkQ.h>
#include <SWCase.h>
#include <SWSortField.h>
#include <SWField.h>
#include <SWWorkItem.h>
#include <SWCriteriaWI.h>
#include <SWException.h>

int main(int argc, char* argv[])
{
    SWEnterprise *pEnterprise;
    SWNodeInfo *pNodeInfo;
    SWEntUser *pEntUser;
    SWNode *pNode;
    SWWorkQ *pWorkQ;
    SWUser *pUser;
    SWXList *pWorkItemsX;
    SWWorkItem *pWorkItem;
    SWSortField *pSortField;
    SWSortField **SortFields;
    SWField *pField;
    SWCriteriaWI *pCriteriaWI;
    SWFieldType fldType;

    int i, j, cnt, blksize;
    char *pTag = NULL;
    char *pKey = NULL;
    char *pXListId = NULL;
    char txtField[256];
    char *FieldNames[2];
    double NumValue;
    SWSortField *SortFldArray[1];

    try {
        printf( "<@@@@@@@@@@@@@@@@@@@@ SAMPLE XLIST START @@@@@@@@@@@@@@@@@@>\n");
        pEnterprise = new SWEnterprise();
        pEnterprise->setBroadcast(false);
        pNodeInfo = pEnterprise->makeNodeInfo("swdoug2", "doug1",
            "10.20.30.108", 3908);
    }
}

```

```

// Create a Enterprise user on represent swadmin
    pEntUser = pEnterprise->createEntUser("swadmin");

// Login swadmin
    pNode = pEntUser->login(pNodeInfo->getKey(), "staffware");
// All work will be done based on scope (i.e. permissions) of logged on user
    pUser = pNode->getLoggedInUser();

    pWorkQ = new SWWorkQ();
    pWorkQ = pNode->makeWorkQ(pWorkQ, "swadmin", true, "doug1");

// Adding strings & objects to XList Criteria
    printf("<=== Adding strings & objects to a XList Criteria ===>\n");
    pWorkItemsX = pWorkQ->getWorkItemsX();
    pWorkItem = (SWWorkItem *) pWorkItemsX->item(0);    // msg sent since first access
                                                    // after clear
// No fields returned for Case associated with Workitem
    printf("Field Count (for workitem:%s) = %d\n", pWorkItem->getKey(),
        pWorkItem->getCase()->getFields()->count());
    FieldNames[0] = "SW_CASEENUM";    // return Staffware Case Number Field
    FieldNames[1] = "TESTPROFIELD3";    // return User defined Field
    pCriteriaWI = (SWCriteriaWI *) pWorkItemsX->getCriteria();
    pCriteriaWI->setCaseFieldNames(2, FieldNames);    //replaces existing array of
                                                    //fieldnames

// Enable WorkItemsX to be sorted by Casenumber

    printf("<=== Show list of preset/default sortfields (ie see std client) ===>\n");
    cnt = pCriteriaWI->getSortFields(SortFields);
    i = 0;
    while( i < cnt) { // Note there are sortfields present by default
        pSortField = SortFields[i];
        printf("Sortfield Field Name = %s\n", pSortField->getFieldName());
        i++;
    }
// Create and configure Sortfield
    pSortField = new SWSortField( "SW_CASEENUM", false, swNumericSort);

// Add SortField
    SortFldArray[0] = pSortField;
    pCriteriaWI->setSortFields(1, SortFldArray);
    printf("<=== List sortfields after configuring criteria on Xlist ===>\n");
    delete pSortField; // sortfield added so free my copy.
    pSortField = NULL;

    i = 0;
    cnt = pCriteriaWI->getSortFields(SortFields);
    while (i < cnt) {
        pSortField = SortFields[i];
        printf("Sortfield Field Name = %s\n", pSortField->getFieldName());
        i++;
    }

    printf("<=== First 25 workitems after adding setting Xlist Criteria ===>\n");
    printf("Total number of workitems on xlist on server = %d\n", pWorkItemsX->getItemCount());

```

```

// Accessing WorkItemsX in XList
// Get only 1st 25 back
if (pWorkItemsX->getItemCount() > 25) {
    cnt = 25; // if more than 25 only loop through 1st 25
}
else {
    cnt = pWorkItemsX->getItemCount(); // if less than 25 loop through all of them
}
i = 0;
while (i < cnt) {
    pWorkItem = (SWWorkItem *) pWorkItemsX->item(i);
    printf("WorkItem Key= %s, CaseNum = %d, Fields returned = %d\n", pWorkItem->getKey(),
        pWorkItem->getCase()->getCaseNumber(),
        pWorkItem->getCase()->getFields()->count());
    j = 0;
    while ( pWorkItem->getCase()->getFields()->isEOL() == false) {
        pField = (SWField *) pWorkItem->getCase()->getFields()->item(j);
        FldType =pField->getType();
        switch(FldType) {
            case swNumericAttr:
                pField->getValue(NumValue);
                printf("    fieldName = %s, Field Value = %d\n", pField->getName(), NumValue);
                break;

            case swTextAttr:
                pField->getValue(txtField, sizeof(txtField));
                printf("    fieldName = %s, Field Value = %s\n", pField->getName(), txtField);
                break;
            default:
                printf("<==== Unexpect Field type ..sample procs had ONLY numeric or text fields =====>\n");
        }
        j++;
    }
    i++;
}
pWorkItemsX->clear(); // Workaround for CR10994
blksize = 5;
pWorkItemsX->rebuild( blksize); // Causes new message to be sent to server so
//criteria takes effect
printf("<=== First 25 workitems after Rebuild with new block size (blk size = 5)===>\n");
printf("Total number of workitems on xlist on server = %d\n", pWorkItemsX->getItemCount());
cnt = 0;
i = 0;
if (pWorkItemsX->getItemCount() > 25)
    cnt = 25;
else
    cnt = pWorkItemsX->getItemCount();
while (i < cnt) {
    pWorkItem = (SWWorkItem *) pWorkItemsX->item(cnt);
    printf("WorkItem Key= %s, CaseNum = %d, Fields returned = %d\n", pWorkItem->getKey(),
        pWorkItem->getCase()->getCaseNumber(),
        pWorkItem->getCase()->getFields()->count());
    j = 0;
    while (pWorkItem->getCase()->getFields()->isEOL() == false) {
        pField = (SWField *) pWorkItem->getCase()->getFields()->item(j);
        FldType =pField->getType();
        switch(FldType) {
            case swNumericAttr:
                pField->getValue(NumValue);
                printf("    fieldName = %s, Field Value = %d\n", pField->getName(), NumValue);
                break;
            case swTextAttr:
                pField->getValue(txtField, sizeof(txtField));
                printf("    fieldName = %s, Field Value = %s\n", pField->getName(), txtField);
                break;
        }
    }
    i++;
}

```

```

        default:
            printf("<==== Unexpect Field type ..sample procs had ONLY numeric or text
                fields =====>\n");
        }
        j++;
    }
    i++;
    if (cnt%blksize == 0) { // minimize memory use on client
        printf("  XList clear  \n");
        pWorkItemsX->clear(); // the block of 5 since already displayed info
    }
}

printf("<=== Local workitem count after listing first 25 workitems with clears ===>\n");
printf("  Local number of workitems = %d\n", pWorkItemsX->count());
// Get only WorkItemsX of procedure TestPro4
pCriteriaWI = (SWCriteriaWI *) pWorkItemsX->getCriteria();
pCriteriaWI->setFilterExpression( "SW_PRONAME = \"TestPro4\"");
pWorkItemsX->rebuild();
printf("<=== First 25 workitems after XList Rebuild with filter expression ===>\n");
i = 0;
if (pWorkItemsX->getItemCount() > 25)
    cnt = 25;
else
    cnt = pWorkItemsX->getItemCount();
while (i < cnt) {
    pWorkItem = (SWWorkItem *) pWorkItemsX->item(i);
    printf("WorkItem Key= %s, CaseNum = %d, Fields returned = %d\n", pWorkItem->getKey(),
        pWorkItem->getCase()->getCaseNumber(),
        pWorkItem->getCase()->getFields()->count());
    j = 0;
    while (pWorkItem->getCase()->getFields()->isEOL() == false) {
        pField = (SWField *) pWorkItem->getCase()->getFields()->item(j);
        FldType = pField->getType();
        switch(FldType) {
            case swNumericAttr:
                pField->getValue(NumValue);
                printf("    FieldName = %s, Field Value = %d\n", pField->getName(), NumValue);
                break;

            case swTextAttr:
                pField->getValue(txtField, sizeof(txtField));
                printf("    FieldName = %s, Field Value = %s\n", pField->getName(), txtField);
                break;

            default:
                printf("<==== Unexpect Field type ..sample procs had ONLY numeric or text
                    fields =====>\n");
        }
        j++;
    }
    i++;
}

pWorkItem = (SWWorkItem *) pWorkItemsX->item(2);
pTag = pWorkItem->getTag(); // Save tag from workitem for accessing item example below
pKey = pWorkItem->getKey(); // Save tag from workitem for accessing item example below

```

```

//Accessing item in SWXList with ItemByKey
printf("Show properties of workitem returned with ItemByKey\n");
pWorkItem = (SWorkItem *) pWorkItemsX->itemByKey(pKey);
printf("Tag = %s\n", pWorkItem->getTag());
printf("MailId = %s\n", pWorkItem->getMailID());
printf("Case Number = %d\n", pWorkItem->getCase()->getCaseNumber());
printf("Procedure Name = %s\n", pWorkItem->getCase()->getProcName());
printf("Case Description = %s\n", pWorkItem->getCase()->getDescription());

printf("Show properties of workitem returned using MakeWorkItemByTag\n");
SWorkItem *pWorkItem1 = new SWorkItem();
pWorkItem1 = pNode->makeWorkItemByTag(pWorkItem1, pTag); // returns same workitem as one
// returned with ItemByKey above
printf("Tag = %s\n", pWorkItem1->getTag());
printf("MailId = %s\n", pWorkItem1->getMailID());
printf("Case Number = %d\n", pWorkItem1->getCase()->getCaseNumber());
printf("Procedure Name = %s\n", pWorkItem1->getCase()->getProcName());
printf("Case Description = %s\n", pWorkItem1->getCase()->getDescription());
delete pWorkItem1;

// Create a XList to persist for future use
pWorkItemsX = new SWXList();
pWorkItemsX = pWorkQ->makeXListItems(pWorkItemsX, 10); // will return 10 items per block
printf("Number of Items in Xlist= %d\n", pWorkItemsX->getItemCount());
printf("Display the tag of first 5 workitems in XList\n");
i = 0;
if (pWorkItemsX->getItemCount() > 5)
    cnt = 5;
else
    cnt = pWorkItemsX->getItemCount();
while (i < cnt) {
    pWorkItem = (SWorkItem *) pWorkItemsX->item(i);
    printf(" WorkItem tag = %s\n", pWorkItem->getTag());
    i++;
}
printf(" Number of items on Xlist after getting first 5 = %d\n", pWorkItemsX->count());
printf("Display the tag of last 10 workitems in XList\n");
i = pWorkItemsX->getItemCount() - 10;
while (i < pWorkItemsX->getItemCount()){
    pWorkItem = (SWorkItem *) pWorkItemsX->item(i);
    printf(" WorkItem key = %s\n", pWorkItem->getTag());
    i++;
}
printf("After getting last 10 workitems on XList (blk size = 5)\n");
printf("Number of items on local Xlist = %d\n", pWorkItemsX->count());

// Persist XList
pCriteriaWI = (SCriteriaWI *) pWorkItemsX->getCriteria();
pCriteriaWI->setPersisted(true); // msg sent to server to keep list
pXListId = pCriteriaWI->getPersistenceId();
printf("SWXList Persisted Id = %s\n", pXListId);

delete pWorkItemsX;

// Disconnect swadmin TCP/IP session from node
pEntUser->disconnect(); // ends TCP/IP connection and removes oNode
// from oEnterprise->EntUser->LoggedInNodes

// Re-connect (ie logon)
pNode = pEntUser->login(pNodeInfo->getKey(), "staffware");

```

```

// Re-attach to persisted XList
pWorkItemsX = new SWXList();
pWorkItemsX = pNode->getXList(pWorkItemsX, pXListId);
printf("Retrieve persisted XList: item count = %d\n", pWorkItemsX->getItemCount());

printf("Before getting last 10 workitems on persisted XList (def blk size = 20)\n");
printf("Display the workitem tags\n");
i = pWorkItemsX->getItemCount() - 10;
while (i < pWorkItemsX->getItemCount()) {
    pWorkItem = (SWWorkItem *) pWorkItemsX->item(i);
    printf(" WorkItem key = %s\n", pWorkItem->getTag());
    i++;
}
printf("After getting last 10 workitems on persisted XList (def blk size = 20)\n");
printf("Number of items on local Xlist = %d\n", pWorkItemsX->count());

delete pWorkItemsX;
delete pWorkQ;
delete pEnterprise;
}
catch(SWException err) {
    printf("Error Code = %d Description = %s\n\n", err.Number, err.Description);
}
catch(...) {
    printf("Unexpected Error\n");
}

return 0;
}

```

Resulting Output

VB Test Output

```

<@@@@@@@@@@@@@@@@ SAMPLE XLIST START @@@@@@@@@@@@@@@@@>
<=== Adding strings & objects to a XList Criteria ===>
Field Count (for workitem:swadmin@dougl|9420803) = 0
<=== Show list of preset/default sortfields (ie see std client) ===>
Sortfield Field Name = SW_HOSTNAME
Sortfield Field Name = SW_PRONAME
Sortfield Field Name = SW_CASENUM
Sortfield Field Name = SW_STEPNAME
<=== List sortfields after configuring criteria on Xlist ===>
Sortfield Field Name = SW_CASENUM
<=== First 25 workitems after adding setting Xlist Criteria ===>
Total number of workitems on xlist on server = 1023
WorkItem Key= swadmin@dougl|9420803, CaseNum = 1, Fields returned = 0
WorkItem Key= swadmin@dougl|9416707, CaseNum = 2, Fields returned = 0
WorkItem Key= swadmin@dougl|9412611, CaseNum = 3, Fields returned = 0
WorkItem Key= swadmin@dougl|9408515, CaseNum = 4, Fields returned = 0
WorkItem Key= swadmin@dougl|9404419, CaseNum = 5, Fields returned = 0
WorkItem Key= swadmin@dougl|9400323, CaseNum = 6, Fields returned = 0
WorkItem Key= swadmin@dougl|9396227, CaseNum = 7, Fields returned = 0
WorkItem Key= swadmin@dougl|9392131, CaseNum = 8, Fields returned = 0
WorkItem Key= swadmin@dougl|9388035, CaseNum = 9, Fields returned = 0
WorkItem Key= swadmin@dougl|9383939, CaseNum = 10, Fields returned = 0
WorkItem Key= swadmin@dougl|9379843, CaseNum = 11, Fields returned = 0
WorkItem Key= swadmin@dougl|9375747, CaseNum = 12, Fields returned = 0
WorkItem Key= swadmin@dougl|9371651, CaseNum = 13, Fields returned = 0
WorkItem Key= swadmin@dougl|9367555, CaseNum = 14, Fields returned = 0
WorkItem Key= swadmin@dougl|9363459, CaseNum = 15, Fields returned = 0
WorkItem Key= swadmin@dougl|9359363, CaseNum = 16, Fields returned = 0
WorkItem Key= swadmin@dougl|9355267, CaseNum = 17, Fields returned = 0

```

```
WorkItem Key= swadmin@dougl|9351171, CaseNum = 18, Fields returned = 0
WorkItem Key= swadmin@dougl|9347075, CaseNum = 19, Fields returned = 0
WorkItem Key= swadmin@dougl|9342979, CaseNum = 20, Fields returned = 0
WorkItem Key= swadmin@dougl|9338883, CaseNum = 21, Fields returned = 0
WorkItem Key= swadmin@dougl|9334787, CaseNum = 22, Fields returned = 0
WorkItem Key= swadmin@dougl|9330691, CaseNum = 23, Fields returned = 0
WorkItem Key= swadmin@dougl|9326595, CaseNum = 24, Fields returned = 0
WorkItem Key= swadmin@dougl|9322499, CaseNum = 25, Fields returned = 0
<=== First 25 workitems after Rebuild with new block size (blk size = 5)===>
Total number of workitems on xlist on server = 1023
WorkItem Key= swadmin@dougl|5234690, CaseNum = 1034, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5238786, CaseNum = 1033, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5242882, CaseNum = 1032, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5246978, CaseNum = 1031, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5251074, CaseNum = 1030, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
XList clear
WorkItem Key= swadmin@dougl|5255170, CaseNum = 1029, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5259266, CaseNum = 1028, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5263362, CaseNum = 1027, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5267458, CaseNum = 1026, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5271554, CaseNum = 1025, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
XList clear
WorkItem Key= swadmin@dougl|5275650, CaseNum = 1024, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5279746, CaseNum = 1023, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5283842, CaseNum = 1022, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5287938, CaseNum = 1021, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5292034, CaseNum = 1020, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
XList clear
WorkItem Key= swadmin@dougl|5296130, CaseNum = 1019, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5300226, CaseNum = 1018, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
  FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5304322, CaseNum = 1017, Fields returned = 2
  FieldName = SW_CASENUM, Field Value =
```

```

    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5308418, CaseNum = 1016, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5312514, CaseNum = 1015, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
XList clear
WorkItem Key= swadmin@dougl|5316610, CaseNum = 1014, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5320706, CaseNum = 1013, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5324802, CaseNum = 1012, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5328898, CaseNum = 1011, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
WorkItem Key= swadmin@dougl|5332994, CaseNum = 1010, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
XList clear
WorkItem Key= swadmin@dougl|5337090, CaseNum = 1009, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value =
<=== Local workitem count after listing first 25 workitems with clears ===>
    Local number of workitems = 5
<=== First 25 workitems after XList Rebuild with filter expression ===>
WorkItem Key= swadmin@dougl|7012356, CaseNum = 6, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value = Rubbish
WorkItem Key= swadmin@dougl|7016452, CaseNum = 5, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value = Primo
WorkItem Key= swadmin@dougl|7020547, CaseNum = 4, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value = fair
WorkItem Key= swadmin@dougl|7024646, CaseNum = 3, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value = poor
WorkItem Key= swadmin@dougl|7028741, CaseNum = 2, Fields returned = 2
    FieldName = SW_CASENUM, Field Value =
    FieldName = TESTPROFIELD3, Field Value = excellent
Show properties of workitem returned with ItemByKey
Tag = dougl|TESTPRO4|swadmin|R|4|7020547
MailId = 7020547
Case Number = 4
Procedure Name = TESTPRO4
Case Description = Third Test Case
Show properties of workitem returned using MakeWorkItemByTag
Tag = dougl|TESTPRO4|swadmin|R|4|7020547
MailId = 7020547
Case Number = 4
Procedure Name = TESTPRO4
Case Description = Third Test Case
Number of Items in Xlist= 1023
Display the tag of first 5 workitems in XList
WorkItem key = dougl|TESTPRO1|swadmin|R|1|9420803
WorkItem key = dougl|TESTPRO1|swadmin|R|2|9416707
WorkItem key = dougl|TESTPRO1|swadmin|R|3|9412611
WorkItem key = dougl|TESTPRO1|swadmin|R|4|9408515
WorkItem key = dougl|TESTPRO1|swadmin|R|5|9404419
Number of items on Xlist after getting first 510
Display the tag of last 10 workitems in XList

```

```
WorkItem key = dougl|TESTPRO1|swadmin|R|1030|5251074
WorkItem key = dougl|TESTPRO1|swadmin|R|1031|5246978
WorkItem key = dougl|TESTPRO1|swadmin|R|1032|5242882
WorkItem key = dougl|TESTPRO1|swadmin|R|1033|5238786
WorkItem key = dougl|TESTPRO1|swadmin|R|1034|5234690
WorkItem key = dougl|TESTPRO4|swadmin|R|2|7028741
WorkItem key = dougl|TESTPRO4|swadmin|R|3|7024646
WorkItem key = dougl|TESTPRO4|swadmin|R|4|7020547
WorkItem key = dougl|TESTPRO4|swadmin|R|5|7016452
WorkItem key = dougl|TESTPRO4|swadmin|R|6|7012356
After getting last 10 workitems on XList (blk size = 5)
Number of items on local Xlist = 23
SWXList Persisted Id = swadmin;swadmin@dougl|R;R;U;L;65539;2417428578
Retrieve persisted XList: item count = 1023
Before getting last 10 workitems on persisted XList (def blk size = 20)
Display the workitem tags
WorkItem key = dougl|TESTPRO1|swadmin|R|1030|5251074
WorkItem key = dougl|TESTPRO1|swadmin|R|1031|5246978
WorkItem key = dougl|TESTPRO1|swadmin|R|1032|5242882
WorkItem key = dougl|TESTPRO1|swadmin|R|1033|5238786
WorkItem key = dougl|TESTPRO1|swadmin|R|1034|5234690
WorkItem key = dougl|TESTPRO4|swadmin|R|2|7028741
WorkItem key = dougl|TESTPRO4|swadmin|R|3|7024646
WorkItem key = dougl|TESTPRO4|swadmin|R|4|7020547
WorkItem key = dougl|TESTPRO4|swadmin|R|5|7016452
WorkItem key = dougl|TESTPRO4|swadmin|R|6|7012356
After getting last 10 workitems on persisted XList (def blk size = 20)
Number of items on local Xlist = 23
```

Index

Symbols

- '*' (asterisk), in REs 120
- '*' character 148, 173
- '^' character 147, 172
- '?' (question mark), in REs 120
- '.' (period), in REs 148, 173
- '\$' character 148, 173
- "/usr/lib/dld.sl exists - can't open shared library"
305
- &ALL& 87, 277
- %DESC 242, 246
- %USER 242, 246
- \$\$WDIR environment variable 6
- \$undeliv work queue 198

A

- Abort, transaction control step 266
- Access authority 221
- Access is denied 302
- Accessing
 - items in an SWList 61
 - items in an SWView 68
 - items in SWLocLists 64
 - work items 190
- Action property 242
- Activating/deactivating
 - client log 284
- Active
 - Server Pages (ASP), debugging 297
- ActiveCnt property 239
- Activity monitoring 37
- Add method 63
- AddAuditEntry method 246
- Adding
 - entries to the audit trail 246
 - entries to the client log 288
 - objects/strings to local lists 63
 - users to a group 219
 - work queue supervisors 208
- Additional
 - views, creating 70
 - XLists, creating 75
- AddNode method 20, 25
- AddNodeEx method 20, 25
- Addressee of work item 113, 140, 165
- AddSupervisors method 208
- AddUsers method 219

- ADMIN access 221
- Admin name 36
- Administering users 214
- "An Exception of Type java.lang.UnsatisfiedLinkError was not handled" 298
- Anonymous logins 35
- Array fields 94
- Arrival date and time 113, 140, 165, 182
- ASP, debugging 297
- AssignAttribute method 223
- Asterisk, in REs 148, 173
- Asynchronous transmission 57
- Attachments 93
- Attributes 221
 - creating 223
 - deleting 224
 - modifying 224
- Audit
 - case data 239
 - data
 - getting when filtering cases 111, 138, 163
 - including in an SWView 72
 - including in an SWXList 83
 - permission 225
 - steps, filtering 100, 102, 128, 130, 154, 156
 - trail 239
 - trail messages, user-defined 246
- audit.mes file 239
- AuditFilterExpr property 100, 102, 128, 130, 154, 156, 245
- AuditProcs property 41, 225, 240
- AuditSteps property 242
- auditusr.mes file 240, 246
- "Authentication Request Failed" 298
- Authority, user 227
- Auto steps 6
- Auto-discovery broadcast 22
- AutoFwds property 200

B

- Background process 6
- Binding 279
- Boolean 17, 18
- Broadcast
 - Interval 23
 - Port Number 23

-
- BSTR strings 56, 61, 64, 81
 - "ByTag" methods 272
 - C
 - C++
 - examples 306
 - C++ objects 7
 - CancelRedirection method 201
 - Case 1
 - administration authority 225, 227, 240
 - closed, make active 268
 - closing 268
 - data 91, 197
 - auditing 239
 - filtering on 117, 144, 169
 - how it affects efficiency 277
 - setting 92
 - description 113, 140, 165, 182, 230
 - family 254
 - filtering 101, 129, 155
 - indexing 279
 - management 230
 - number, obtaining 233
 - prediction 247
 - purging 269
 - reference number 113, 140, 166, 182
 - start authority 226, 234
 - Start function 226, 234
 - starting 230
 - status, active or closed 268
 - suspending 254
 - Case data
 - getting, impact of 105, 133, 159
 - Case Data Queue Parameter fields
 - defined 93
 - filtering on 117, 144, 169
 - sorting on 184
 - Case property 212
 - CaseCnt property 239
 - CaseDataQParamDefs property 117, 144, 169, 184
 - CaseDataQParams property 117, 145, 170, 184
 - CaseFieldNames property 105, 133, 159, 277
 - CaseNumber property 233
 - CasePredictQParams property 251
 - Cases
 - alternate view/XList 237
 - default view/XList 236
 - determining number of 238
 - property 15, 65, 236
 - CasesX property 15, 74, 236
 - Categories
 - property 285
 - CDQPNames property 117, 145, 170, 184
 - ChangeAttribute method 223, 224
 - ChangeParticipation method 205
 - ChangePassword method 217
 - ChangeRedirection method 201
 - Changing user's password 217
 - Character encoding 291
 - CheckOSUser 217
 - CIList 83
 - ClassId 56
 - Clear
 - method 60, 78
 - views and lists 279
 - ClearDefCriteria method 124, 151, 176, 187
 - Client log 280
 - activating/deactivating 284
 - adding entries to 288
 - displaying memory information 288
 - filtering 285
 - name and location 284
 - resetting 288
 - setting size 288
 - testing 289
 - Client/server 3
 - ClientLog property 282
 - Client-to-Server Communications 7
 - CloseByCriteria method 123, 150, 176
 - ClosedCnt property 239
 - Closing cases 268
 - based on filter criteria 123, 150, 176
 - Cnt (in property name) 17
 - Code examples 306
 - COM objects 7
 - Commit and Concede, transaction control step 266
 - Commit and Continue, transaction control step 266
 - Common Interface for List Objects 83
 - Communications
 - asynchronous/synchronous 57
 - optimizing 279
 - Comparison operators 105, 109, 133, 136, 159, 162
 - Connection ID, database 36
 - Constructing
 - SWDate objects 202, 205
 - Controlling resources, SWXLists 77
-

-
- Conversions, data type when filtering 116, 143, 168
 - Count property 278
 - SWList 60
 - SWList/SWView 58
 - SWLocList 63
 - SWView 67, 100, 128, 154, 192
 - SWXList 80, 102, 130, 156, 193, 238
 - CreateAttribute method 223
 - CreateAutoFwd method 200
 - CreateEntUsers method 31
 - CreateGroup method 218
 - CreateObject 279
 - CreateParticipation method 205
 - CreateRole method 220
 - CreateUser method 216
 - Creating
 - additional views 70
 - additional XLists 75
 - attribute 223
 - redirection schedule 201
 - role 220
 - Staffware user 216
 - user group 218
 - Criteria property 101, 129, 155
 - Custom audit-trail messages 246
 - D**
 - Data type
 - conversions, filtering 116, 143, 168
 - sorting by 188
 - used in filtering 116, 142, 168
 - Database case filtering 98, 126, 152
 - Database configuration 36
 - Date
 - data type 116, 142, 168
 - format 97
 - in filter expressions 122, 149, 175
 - object 202, 205
 - DateCreated property 45
 - DateModified property 45
 - DateOnly 205
 - DateReleased property 45
 - DateTime 202
 - data type 116, 142, 168
 - in filter expressions 122, 149, 175
 - DateWithdrawn property 45
 - DB_Enabled 200, 204
 - DBConnectionAccess 36
 - DCOM 300
 - Deadline 209
 - date and time 113, 140, 166, 182
 - expired flag 113, 140, 166, 182
 - property 209
 - set flag 114, 140, 166, 182
 - DeadlineCnt property 81, 210
 - Default
 - filter criteria 124, 150, 176
 - sort specifications 186
 - Views of Cases and Work Items 65
 - XLists of objects 74
 - Definer, iProcess 41
 - Delete method 63
 - DeleteAttributes method 224
 - DeleteAutoFwd method 200
 - DeleteGraftTask method 263
 - DeleteGroups method 219
 - DeleteRoles method 220
 - Deleting
 - attribute 224
 - objects/strings from local lists 63
 - role 220
 - Staffware user 217
 - user group 219
 - work item on withdrawal 211
 - Directed UDP 20, 25
 - Director 8
 - Directory
 - system 6
 - DisableCategory method 286
 - DisableMessage method 287
 - Disconnect method 35, 271
 - Disconnecting from the SPO Server 271
 - DNS 25
 - Dr. Watson 297
 - Duration property 248
 - Dynamic
 - sub-procedures 48
 - TCP port 21, 27
 - Dynamic sub-procedure call step
 - outstanding 256
 - DynamicSubProcSteps property 256
 - E**
 - EAI step 212
 - EAI step, outstanding 256
 - EAI steps 265
 - EAISteps property 212, 256
 - Early binding 279
 - Efficiency
-

- filtering 277
 - getting case data 277
 - handling large lists 275
 - sorting 182
- Empty 202, 206
 - fields, in filtering 121, 149, 174
- Empty object, in make methods 273
- EnableCategory method 286
- EnableMessage method 287
- Encoding, character 291
- Enterprise user 15, 215
- EntUsers 15, 31
- Environment variables
 - client log 283
- EOErrs.h file 296
- Err object 293
- err.h file 296
- Error
 - constants 292, 295
 - handling
 - C++ 296
 - COM 292
 - Java 294
 - result of processing work item 198
 - trapping in Visual Basic 293
- "Error 2140: An internal Windows NT error occurred" 298, 299, 301, 302
- "Error 5: Access is denied" 302
- "Error Calling CreateDataSource Interface for SWAutoFwdQ" 302
- "Error creating mutex" 303
- "Error in sal_frm_putdata call" 304
- "Error Initializing AutoFwd/QView Database" 302
- Escape characters, in filter expressions 121, 148, 174
- Event publishing 37
- Event step, outstanding 256
- EventSteps property 256
- Examples 306
- Exception Monitor 297
- Exceptions, Java 294
- ExcludeCnt property 76, 80
 - SWView 71, 100, 128, 154, 193, 238
 - SWXList 102, 130, 156
- Executing command
 - on work item keep 197
 - on work item release 197
- Expired deadline 209
- Expressions property 235
- External
 - processes 260
 - work items 212
- External processes 262
- ExternalId property 212
- ExtForm property 212
- ExtProcesses property 262
- ExtProcessNames property 260
- ExtWorkItemId property 212
- F**
- Facility code 292
- Field
 - Staffware 86
 - types, validating 89, 90
- Fields
 - array 94
 - property 86, 277
 - System 94
- FilterExpression property 100, 101, 128, 129, 154, 155
- Filtering 99, 127, 153
 - audit steps 100, 102, 128, 130, 154, 156, 245
 - cases in database 98, 126, 152
 - client log 285
 - data types 116, 142, 168
 - expression format 111, 138, 163
 - in an efficient manner 277
 - on case data 117, 144, 169
 - on deadline information 210
 - on empty fields 121, 149, 174
 - on ranges of values 122, 149, 175
 - on SWViews 100, 128, 154
 - on SWXLists 101, 129, 155
 - predicted items 252
 - using system fields 113, 140, 165
 - work items in WIS 98, 126, 152
- Finalization 8
- Finding available nodes 20
- Firewall, using with SPO 30
- FirstDeadline property 209
- Flat file system 6
- For Each construct 278
- Form
 - Staffware 88, 232
- Format, of date 97
- Forwardable work item flag 114, 141, 166, 182
- Forwarding
 - permission 199, 222

-
- work items 199
 - ForwardItem method 199
 - FwdItems property 199
 - G**
 - Garbage collection 8
 - "get" method names 17
 - getActivityPub method 38
 - getDatabaseConfig method 36
 - GetExtWorkItem property 212
 - GetXList method 81
 - GetXListPredict method 81, 249
 - Graft step
 - outstanding 256
 - Graft steps 259
 - GraftExtProcessComp method 260, 262
 - GraftSteps property 256, 261
 - GrantAccess method 203
 - Granting Access 203
 - Group 2
 - user 218
 - work queue 2, 16
 - Groups property 218
 - GroupsX property 74, 218
 - H**
 - Hidden case description 230
 - Hood ID 281
 - Hosting node 2
 - HRESULT 292
 - I**
 - IAP JMS library 37
 - IAPConfigAccess 37
 - ICU conversion libraries 291
 - IDispatch pointers 61, 64, 81
 - IDX_ 95
 - IMPMONITOR command 37
 - Index of the participation schedules 207
 - Indexes, array fields 95
 - Indexing, cases 279
 - InFromFldNames property 50
 - InitialExpr property 196
 - InstanceNumber property 23, 26
 - Internet Information Server (IIS) 297
 - InToFldNames property 50
 - InvalidCnt property 76, 80
 - SWView 72, 100, 128, 154, 193, 238, 239
 - SWXList 102, 130, 156
 - IP address 26
 - iProcess Definer 41
 - iProcess Engine 5
 - IsActive property 268, 284
 - IsArrayField property 94
 - IsAuditAscending property 72, 83, 241
 - IsBroadcast property 22, 25, 27
 - IsChanged property 89
 - IsDeadline property 209
 - IsDeadlineAWD property 209
 - IsDeadlineExp property 209
 - isDefault property 203, 207
 - IsEOL property 278
 - SWList 60
 - SWView 66, 192
 - isExplicit property 203, 207
 - IsGraftOutstanding property 261
 - IsHaltOnSubProc property 49, 263
 - IsHaltOnTemplate property 49, 263
 - IsHaltOnTemplateVer property 49, 263
 - IsIgnoreState property 255
 - IsKeepLocalItems property 77, 78, 276
 - IsKeepOnWithdrawal property 211
 - IsLocked property 195
 - IsLongLock property 195
 - IsMandatory property 53
 - isNoChange property 203, 207
 - IsOrphaned property 198
 - IsOutStanding property 242
 - IsOutstanding property 261
 - IsPersisted property 81
 - IsPrediction property 249
 - IsPublic property 53
 - IsRebuildAll property 62, 70
 - IsRecurseProcPath property 256, 261
 - IsReleasable property 198
 - IsReleased property 190
 - IsSendValue property 89, 196, 197
 - IsShowMemory property 288
 - IsSuspended property 254
 - IsTaskCntSet property 261
 - IsUndelivered property 198
 - IsUnopen property 196
 - IsUserPwdExp property 33
 - IsWaitForAll property 57
 - IsWithAuditData property 46, 72, 83, 111, 138, 163, 241
 - IsWorkQReleased property 190
 - Item method
 - SWList 60, 61
 - SWLocList 64
 - SWView 67, 68
-

-
- ItemByKey method
 - object keys 84
 - SWList 60, 61
 - SWLocList 64
 - SWView 67, 68
 - ItemCount property 76, 80, 102, 130, 156, 193, 238
 - ItemsPerBlock property 73, 75
 - J**
 - Java
 - examples 306
 - exceptions 294
 - Message Service (JMS) 37
 - Native Interface (JNI) 298
 - objects 7
 - JumpTo method 255
 - JumpToStepName 243
 - K**
 - KeepExpr property 197
 - Keeping
 - multiple items 278
 - multiple work items 296
 - start step 231
 - the start step 92
 - work item 2, 196
 - Key property 84, 190
 - L**
 - LAN 21
 - segment 20
 - LANGUAGE attribute 221
 - Large lists, how to handle 275
 - LastError property 198, 296
 - LastUpdateUser property 45
 - Late binding 279
 - Latency 23, 271
 - LD_LIBRARY_PATH 305
 - LIBPATH 305
 - Limiting number of items in a view 72
 - Lists, using 55
 - Local
 - host file 25
 - lists 55
 - LockedBy property 196
 - Locker of the work item 114, 141, 166, 183
 - Locking
 - multiple work items 278, 296
 - work items 194
 - LockItem method 88, 194
 - LockItemMarkings method 88, 194
 - LockItems method 88, 194
 - LockItemsEx method 88, 194
 - LockItemsMarkings method 88, 194
 - Log
 - level, client log 285
 - method 288
 - LogDirectory property 284
 - LoggedInNodes 15, 32
 - Logging
 - client 280
 - in and out of Staffware 32
 - Logical operators 105, 109, 133, 136, 159, 162
 - LogId property 284
 - Login
 - anonymous 35
 - daemon 6
 - failure 33
 - long-lived 20, 21
 - method 32, 270
 - short-lived 20, 21
 - LogLevel
 - property 285
 - Logout method 270
 - Long lock 305
 - Long-lived logins 20, 21
 - Looping through items in a list 278
 - M**
 - Major version number, procedures 42
 - "Make" methods (for stateless programming) 271
 - MakeNodeInfo method 20, 26
 - MakeNodeInfoEx method 20, 26
 - MakeProc method 44
 - MakeProcByStatus method 44
 - MakeStep method 45
 - MakeViewCases
 - implied sort order 188
 - method 70, 237, 238
 - MakeViewItems
 - implied sort order 188
 - method 70, 191
 - MakeViewItemsByTag method 191
 - MakeWorkItemByTagEx method 86
 - MakeWorkItemEx method 86
 - MakeXListCases method 75
 - MakeXListItems method 75, 81, 192
 - MakeXListItemsEx method 75, 81, 86, 192
 - MakeXListPredict method 81, 249
 - MANAGER access 221
 - Managing work queues 189
-

- Manual configuration 20
- Markings 88
 - populating 194
 - re-populating 196
 - types 232
 - user-created 90
 - validating 232
- MaxCnt property 72, 79
- MaxSize property 288
- Memo fields 93
- Memory
 - freeing 78, 276, 279
 - Information, in the client log 288
- Menu property 221
- MENUNAME attribute 221
- MenuName property 221
- Message
 - Event Request (MER) 37
 - property 242
 - threads 298
- Messages
 - property 287
- MessageWaitTime 93, 290
- Methods 17
- Microsoft
 - Access Driver 301, 303
 - RDO 2.0 drivers 303
- Minor version number, procedures 42
- Model 43, 190
- Modeler 1
- Monitoring activities 37
- MOVESYSINFO function 215
- Multi-byte encoding 291
- Multiple
 - instances of SPO Server 23, 26
 - items, locking/keeping/releasing 278
 - logins 302
 - views, creating 70
 - work item operation errors 296
- Mutex 303

- N**
- Names (in property name) 17
- Naming conventions 17
- New work item flag 115, 142, 167, 183
- Node 2
- NodeInfos property 15, 22, 26
- Nodes
 - finding available 20
- Normal step, outstanding 256
- NotEmpty 202, 206

- Number of
 - items in an XList 80, 193
 - objects in an SWList or SWView 58
 - work items in a work queue 192
- Numeric data type 116, 142, 168

- O**
- Object
 - keys 84
 - types 56
- ODBC connection name 36
- On Error function 293
- "One of the items in the array returned an error" 304
- Operating system users 215
- Operators
 - comparison 105, 109, 133, 136, 159, 162
 - logical 105, 109, 133, 136, 159, 162
- Optimizing
 - communications 279
 - SPO applications 275
 - VB application performance 279
- Optional case description 230
- Oracle SID 36
- ORACLE_HOME directories 305
- Orphaned work item 198
- OSUsersX property 74
- OutFromFldNames property 50
- Out-of-the-box client 7
- Outstanding
 - graft items 261
 - sub-procedures 50
- OutstandingItems property 211, 256
- OutToFldNames property 50
- OverMaxCnt property 72

- P**
- Pack data/file 91
- Packfile data 196
- Page File Usage 288
- Parallel steps 92
- Parameter
 - templates 49, 263
- Participation access to work queue 204
- Password
 - database user 36
- Password checking 33, 217
- Permission
 - to audit 225
- Persisted XLists 81, 274
- PersistenceId property 81

-
- Persisting
 - filter criteria 124, 150, 176
 - PollCnt property 23, 24
 - Populating
 - SWLists 59
 - SWViews 66
 - SWXLists
 - Cases 79
 - groups, users, or OSUsers 80
 - Precedence, sub-procedure 233
 - PredictCase method 249
 - Predicting cases 247
 - Prediction flag 249
 - PredictType property 249
 - Priority of work item 81, 114, 141, 166, 183
 - ProcAudits property 46
 - Procedure 1, 15, 41
 - audit trail 46
 - name 114, 141, 166, 183
 - status 43
 - version control 42
 - Process
 - Engine 5
 - Invoker 6
 - ProcGroups property 44
 - ProcMajorVer property 42
 - ProcMinorVer property 42
 - ProcPath property 258
 - Procs property 41
 - ProcVersions property 44
 - PRODEF access 221
 - Properties 17
 - Provider, database 36
 - Pstaffli 6
 - pthread.dll 304
 - Public steps, fields 53
 - PublicFields property 53
 - PublicStepDesc property 53
 - PublicStepURL property 53
 - Publishing events 37
 - PurgeByCriteria method 123, 150, 176
 - Purging cases 269
 - based on filter criteria 123, 150, 176
 - Q**
 - QSUPERVISOR attribute 208, 221
 - Queue not found error 190
 - R**
 - Range filtering 122, 149, 175
 - Raw data buffers
 - SWList 59
 - SWView 66
 - SWXList of cases 79
 - Rebuild method
 - SWList 60
 - SWView 67
 - SWXList 75
 - using unnecessarily 276
 - Rebuilding
 - an SWView 69
 - subordinate lists 62, 69, 70
 - ReceiveLog property 282
 - Redirecting work items 200
 - Redirection property 201
 - Registry
 - client log 283
 - Regular expressions 120, 147, 172
 - Releasable
 - work item 198
 - work item flag 114, 141, 166, 183
 - Released work queue 190
 - ReleaseExpr property 197
 - ReleaseItem method 212
 - ReleaseItem method (for EAI steps) 212
 - Releasing
 - multiple work items 278, 296
 - start step 92, 198, 231
 - work item 2, 197
 - RemoveParticipation method 205
 - RemoveSupervisors method 208
 - RemoveUsers method 219
 - Removing
 - users from a group 219
 - work queue supervisors 208
 - RequestId property 212
 - Required case description 230
 - ResetLog method 288
 - Resetting
 - case counter 269
 - the client log 288
 - Resource control, SWXLists 77
 - ResumedDescription parameter 243
 - ResumedStepName parameter 243
 - RetryDelay property 267
 - RetryTime property 267
 - Return status, graft step 262
 - ReturnStatus property 49, 262
 - RevokeAccess method 203
 - Role
 - creating 220
 - deleting 220
-

- RoleNames property 235
- root user
 - access to system directory 6
- Router 24
- RPC
 - Server 6
- S**
- SAL session
 - starting 270
- Script, running when
 - keeping work item 197
 - locking work item 196
 - releasing work item 197
- Semaphores 303
- SEOPasswordRequired parameter 33
- Server
 - multiple on one machine 27
 - service 24, 299, 301, 302
- Service, server 24, 299, 301, 302
- services file 28, 29
- "set" method names 17
- setActivityPub method 38
- SetCaseData method 91
- setDate method 207
- setDateTime method 203
- setDefault method 203, 207
- SetDefCriteria method 124, 150, 176, 186
- SetDefCriteriaEx method 124, 150, 176, 186
- SetGraftTaskCnt method 261
- setNoChange method 203, 207
- SetState method 254
- setTime method 207
- Setting case data 92
- SHLIB_PATH 305
- Short-lived logins 20, 21
- SimulateCase method 250
- Single-byte encoding 291
- Single-parameter methods (stateless) 272
- Size of
 - client log 288
- Sort criteria 107, 111, 134, 138, 160, 163
- SortAsType property 188
- SortFields property 178, 179
- Sorting 178
 - data types 188
 - in an efficient manner 182
 - on case data 183, 184
 - on SWViews 178
 - on SWXLists 179
 - predicted items 252
 - setting default specifications 186
 - system fields 182
- SORTMAIL attribute 222
- SPO
 - Server 15
- SQL select statement 98, 126, 152
- StackSize parameter 157
- staffcfg file 6
- staffli.exe 6
- staffo
 - database table 216
 - file 6, 216
- staffpms file 33, 97, 302
- Staffware
 - form 88, 232
 - user 215
 - User Manager 214
- Start step, keeping/releasing 92, 198, 231
- StartByUserRef property 235
- StartCaseDescription parameter 243
- StartCaseStepName parameter 243
- Started date and time of the case 114, 141, 166, 183
- Starter of the case 114, 141, 166, 183
- StartGraftTask method 260
- StartIndex 52
- StartIndex property 52, 259
- Starting
 - a case 230
 - a graft task 260
 - a SAL session 270
 - sub-procedure 230
- StartProcs property 41, 226, 235
- StartStepName property 230
- Stateless programming 270
- Static TCP port 21, 28
- Status
 - of the case 114, 141, 166, 183
 - property 43, 67, 76, 237, 238
- Step 2
 - Forward 199, 222
 - parallel 92
- StepName property 212
- SubCaseId property 244
- Subordinate lists 62, 70
- Sub-procedure
 - dynamic 48
 - outstanding 50, 256
 - precedence 233

- starting 230
 - using 46
- SubProcName property 47, 48, 260
- SubProcPath property 51, 258
- SubProcStartStep property 47, 48
- SubProcStatus property 260
- SubProcSteps property 256, 261
- SupervisorNames property 208, 221
- SuspendedDescription parameter 243
- SuspendedStepName parameter 243
- Suspending cases 254
- SW_GEN_IDX 96
- SW_NA 121, 149, 174
- SW_QPARAM1 119, 146, 171
- SW_QPARAM1-4 118, 145, 170, 185
- SW_STATUS 268
- SWAccessUserRef 235
- swadmin user 6, 221, 227
- SWAuditActionType 38, 242, 254
- SWAuditStep 242
- SWAWorkQ 201
- SWCase 15
- SWCaseDataQParamDef 117, 144, 169, 184
- SWCasePredictQParam 251
- SWCaseStateType 254
- swChanged 67, 71, 76, 191, 237, 238
- SWCIList 83
- SWClientErrorType 292, 295
- SWConditionPredictType 249
- SWCriteriaC 101, 129, 155, 179
- SWCriteriaP 101, 129, 155, 179
- SWCriteriaWI 101, 129, 155, 179
- SWDatabaseConfig 36
- SWDate 14, 202, 205, 207
- SWDeadline 209
- SWDIR, system directory 6
- SWDurationType 248
- SWDurationValue 248
- swDynamicReleased 243
- SWDynamicSubProcStep 256
- swDynamicWithdrawn 243
- SWEAISStep 212, 256
- SWEnterprise 14
 - creating 19
 - destructing 19
- SWEntObjDB.exe 200, 203
- SWEntUser 15, 215
 - creating 31
- SWEOCom.dll 279
- swErrBadSubProc 243
- swErrDiffTemplate 243
- swErrDiffTemplateVer 243
- SWEEventStep 256
- SWException object 296
- SWExtProcess 262
- SWExtWorkItem 212
- SWField 86
- SWFMarking 88
- SWFwdItem 199
- SWGraftStep 256, 261
- SWGGroup 16, 218
- swHiddenDesc 230
- swIncomplete 43, 235
- SWIPEConfig 36, 38
- SWIPEConfig object 37
- SWList 55, 59
 - looping through items 278
 - when to rebuild 62
- swLLDupErr 25, 27
- SWLocList 55, 63
- SWLog object 282
- SWLogCategoryType 285
- swLoginFailErr 26, 33
- SWLogMessageType 287
- SWMarking 14, 88
- swModel 43, 190
- SWMonitorList.xsd schema 37, 38
- swNoChange 67, 76
- SWNode 15
- SWNodeInfo 15
- swOptionalDesc 230
- SWOutstandingItem 256
- SWPredictedItem 247
- swpro user 6
- swpro.exe 6
- SWProc 15
- SWProcGroup 44
- SWProcStatusType 43
- SWPublicField 53
- SWQSessionInfo 81
- swReleased 43, 190, 235
- swRequiredDesc 230
- swrpsrv.exe 6
- SWServerErrorType 292, 295
- SWSortField 14, 179
- SWSortType 188
- swStatusOnly 67
- swSubCaseComp 243
- swSubCaseExpired 243
- swSubCaseStart 243
- swSubCaseTerm 243
- swSubCaseWithdrawn 243

-
- SWSubProcPrecedenceType 233
 - SWSubProcStatusType 262
 - SWSubProcStep 51, 256, 261
 - SWTransControlStep 256
 - SWTransControlType 266
 - swUnreleased 43, 190, 235
 - SWUser 16, 216
 - swuser user 6
 - swuser.exe 6
 - SWView 55, 65
 - alternate 70
 - determining if filled 71, 192
 - including audit data 72
 - looping through items 278
 - swWINSOCKErr 22, 25
 - swWithdrawn 43, 235
 - swWithdrawnIncomplete 43
 - SWWorkItem 16
 - SWWorkQ 16
 - swXLChanged 237, 238
 - SWXList 55, 73, 275
 - including audit data 72, 83
 - persisted 81, 274
 - swXLStatusOnly 76
 - SYSTEM account 302
 - System Administrator authority 227
 - System directory 6
 - System fields 94
 - used in filtering 113, 140, 165
 - used in sorting 182
- T**
- Tag property 272
 - TaskCnt property 261
 - TCP
 - name resolution 25
 - port
 - for database 36
 - port number 26, 299
 - TCP/IP communications 3, 7
 - Terminated date and time of the case 115, 141, 167, 183
 - TerminationDescription 243
 - TerminationDescription parameter 243
 - TerminationStepName 243
 - TerminationStepName parameter 243
 - TerminationUser 243
 - TerminationUser parameter 243
 - Test work queue 190
 - Testing the client log 289
 - Text data type 116, 142, 168
 - "The memory could not be "written"" 304
 - Thick client
 - creating SWEnterprise 19
 - logging in and out of 34
 - login 21
 - Third-party application 212
 - TIBCO
 - iProcess Engine 5
 - iProcess Modeler 1
 - iProcess Objects Director 8
 - iProcess Objects Server 7
 - Process Engine 5
 - Time
 - data type 116, 142, 168
 - in filter expressions 122, 149, 175
 - TimeOnly 206
 - Timeout, server reply 93, 290
 - TISOMultiChar 291
 - TISOUnicodeConverterName 291
 - TNS connection 36
 - Transaction control steps 265
 - TransControlSteps 256
 - TransControlType property 266
 - Transparent Network Substrate (TNS) 36
 - TriggerEvent method 268
 - Event, triggering 253
 - Troubleshooting IIS 297
 - Type property 56, 89
- U**
- UDP broadcast 20
 - UDP_SERVICE_NAME process attribute 23, 24, 25
 - UDPPort property 23, 25
 - UDPServiceName Parameter 23, 24, 25
 - "Unable to locate DLL" 304
 - Undelivered work item 198
 - Undoing
 - multiple work items 296
 - work item changes 195
 - Unlocking
 - a work item 195
 - multiple work items 296
 - Unopened work item flag 115, 142, 167, 183
 - UnopenedCnt property 81, 196
 - Urgent work item flag 115, 142, 167, 183
 - UrgentCnt property 81
 - User 2
 - administration 214
-

- attributes 221
 - authority 227
 - creating 216
 - deleting 217
 - Enterprise 15
 - enterprise 215
 - groups 218
 - Message, in client log 281
 - USER access 221
 - User-created markings 90
 - User-defined audit trail messages 240
 - USERFLAGS attribute 222
 - USERINFO 215
 - UserNames property 235
 - Users property 216
 - UsersX property 74, 216
 - UTF-8 291
- V**
- Validating
 - field types 89, 90
 - markings 232
 - Value property 86
 - ValueType property 89
 - Variant 61, 64, 81
 - VB applications, optimizing performance 279
 - vbObjectError 293
 - Version control, procedures 42
 - VersionComment property 45
 - View-only access to work queues 203
 - ViewOnlyQs property 203
 - Views 55, 65
 - ViewUserName property 203
 - Visual
 - Basic examples 306
 - J++ 298
- W**
- WAN 21
 - Web-based environment 271
 - creating SWEnterprise 19
 - logging in and out of 34
 - logins 21
 - Wild card characters, filtering 106, 110, 133, 137, 159, 162
 - WIS work item filtering 98, 126, 152
 - WIS-compatible 113
 - wisrpc.exe 6
 - Withdrawing outstanding items 255
 - Withdrawing work item 209, 211
 - Withdrawing work item on deadline 209
 - Work item 2
 - accessing 190
 - counts 81
 - data 91, 118, 145, 170, 196
 - deadlines 209
 - determining if new (unopened) 196
 - determining who locked 196
 - errors 198
 - external 212
 - filtering 101, 129, 155
 - forwarding 199
 - keeping 196
 - locking 194
 - orphaned 198
 - processing 194
 - releasable 198
 - releasing 197
 - server 6
 - undelivered 198
 - undoing changes 195
 - unlocking 195
 - withdrawing 211
 - "Work Item is not accessible" 305
 - Work queue 2
 - forwarding 199
 - granting access to 203
 - managing 189
 - parameter fields
 - for filtering 118, 145, 170
 - for sorting 185
 - participation 204
 - server 6
 - test/released 190
 - "Work Queue not found" 217
 - Working Set Size 288
 - WorkItems property 16, 65, 191
 - WorkItemsX property 16, 74, 192
 - WorkQs property 16, 189
 - WQParam1-4 properties 119, 146, 171, 186
 - WQParam1Name - 4Name properties 119, 146, 171
 - WQS_WIS_COUNT 6
 - wqsrpc.exe 6
- X**
- XLists 55, 73, 275