

Process System Guide

iPlanet™ Integration Server

Version 3.0

August 2001

Copyright (c) 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Sun, Sun Microsystems, the Sun logo, Java, iPlanet and the iPlanet logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

This product includes software developed by Apache Software Foundation (<http://www.apache.org/>). Copyright (c) 1999 The Apache Software Foundation. All rights reserved.

Federal Acquisitions: Commercial Software – Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright (c) 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats - Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, Java, iPlanet et le logo iPlanet sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Ce produit inclut des logiciels développés par Apache Software Foundation (<http://www.apache.org/>). Copyright (c) 1999 The Apache Software Foundation. Tous droits réservés.

Acquisitions Fédérales: progiciel – Les organisations gouvernementales sont sujettes aux conditions et termes standards d'utilisation.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISÉE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

Contents

List of Figures	13
List of Procedures	15
Preface	19
Product Name Change	19
Audience for This Guide	20
Organization of This Guide	20
Text Conventions	21
Syntax Statements	22
Other Documentation Resources	22
iPlanet Integration Server Documentation	23
Online Help	23
Documentation Roadmap	23
iIS Example Programs	24
Viewing and Searching PDF Files	24
Chapter 1 Introduction: iIS Process Management	27
What Is an iIS Process Engine?	27
Multiple Engine Systems	29
iIS Process Engine Components	31
Minimal Engine Configuration	31
Failover Configuration	33
Full Configuration: Failover and Load Balancing Combined	35
What Does an iIS Process Engine Do?	36
iIS Process Management Tasks	37
Setting up and Maintaining an iIS System	37
Setting up an iPlanet UDS Runtime Environment	38
Installing iIS Process Management Software	38
Setting Up and Maintaining Central Development Repositories	38

iIS Process Management Tasks <i>(continued)</i>	
Managing iIS Process Engines	39
Managing Registration	39
Managing Process Execution	39
iIS Process Management Tools	40
iIS Console	40
Conductor Script Utility	41
Custom System Management Tools	41
Repository Management Tools	41
Dump/Restore Facilities	41
Chapter 2 Setting Up an iIS Process Management System	43
Preparation: Setting up an iPlanet UDS Runtime Environment	44
iIS Process System Components	45
iIS Backbone	47
iIS Process System Software	47
Compiled and Interpreted Engine Components	50
iIS Process System Configuration	51
Setting Up an iIS Process System	55
iIS Process System Setup Procedure	55
The iIS Installation Program	56
Central Server Installation	57
Engine Server Installation	58
Development Client Installation	59
Runtime Client Installation	60
Configuring and Starting Your iIS Engines	60
Maintaining an iIS Process System	61
Adding New Nodes to an iIS System	61
Moving an iIS Engine	62
Creating a New iIS Repository Server	62
Moving an iIS Repository Server	62
Creating a Private iIS Repository	63
Upgrading an iIS System	63
Uninstalling an iIS System	64
Chapter 3 The iIS Console	65
Overview	65
Starting the iIS Console	67
Using the cconsole Command	67

The iIS Console Main Window	68
Main Viewing Panel	69
Menu Bar	70
Mouse Popup Menu	70
Online Help	70
Setting Password Protection for iIS Console	71
Exiting iIS Console	71
Using iIS Console Windows	72
Using the Mouse	72
Window Refresh Behavior	72
Filtering iIS Console Lists	73
Operators	74
Specifying Values	74
Example Filter Expression	75
Sorting iIS Console Lists	76
Using List Views	77
iIS Console Main Window Command Summary	78
Environment Menu	78
View Menu	79
Engine Menu	79
Monitor Menu	80
Help Menu	81
Chapter 4 Managing Engines	83
Production Engines Versus Development Engines	83
Configuring an Engine	85
Engine Component Partitioning	85
Engine Startup Properties	88
Engine Configuration File	90
Customizing Engine Database Schema	93
How to Configure a New Engine	95
Duplicating an Engine Configuration	104
Deleting an Engine Configuration	104
Starting an Engine	105
Governor	105
Engine Unit	106
Primary Engine Unit Startup Phases	108
Database Service	109
How to Start an Engine	110

Reconfiguring an Engine	113
How to Reconfigure an Engine	113
How to Dynamically Modify Database Logging	115
How to Tune Process Execution	116
Monitoring and Changing Engine States	117
Monitoring Engines and Engine Components	118
Monitoring the Engine	118
Monitoring Engine Components	119
Changing Engine States	122
Changing Engine Unit States	122
About Recovering State Information	123
Shutting Down Engine Components	124
Managing an Engine Database	124
Database Management Issues	124
Growth of the Database	125
Failure of the Database	125
Recovering Data	125
Dumping and Restoring Data	126
Dumping Database Tables	126
Restoring Database Tables	131
Dump/Restore Environment Variables	136
Chapter 5 Managing Registrations	137
About Registration	137
What Does Registration Do?	139
Registration in Production Environments	141
What Does Unregistration Do?	142
Engine Registration Manager	142
About Aliases	143
Registering iIS Distributions	144
Registration Order	145
Performing Registrations	145
Upgrading Registrations	147
Unregistering iIS Distributions	147
Registering Aliases	148
Unregistering Aliases	150
Viewing Registrations for an Engine	150
Unregistering a Distribution	151
Monitoring Instances of a Registered Process Definition	152
Performing Application Upgrades	153
Monolithic Upgrades	154
Rolling Upgrades	155

Chapter 6 Managing Process Execution	157
Introduction	157
Engine Sessions	158
Disrupted Sessions	159
Explicitly Suspended Sessions	161
Explicitly Terminated Sessions	161
Activity States	162
Activity Types	165
Process Execution	165
Process Instance Creation	166
Process Instance Execution	167
Activity Creation	167
Activity Execution	168
Activity Termination	171
Process Instance Termination	173
Completed Process Instances	173
Aborted Process Instances	173
Monitoring and Managing Engine Sessions	173
Monitoring the State of a Session	174
Managing Sessions	176
Suspending or Terminating Sessions	176
Sending and Broadcasting Messages to Sessions	177
Monitoring and Managing Process Execution	179
Managing Process Instances	181
Checking the Status of a Process Instance	181
Aborting a Process Instance	184
Managing Activity Instances	185
Checking the Status of an Activity	185
Changing the State of an Activity	186
Managing Activity Queues	188
Checking the Status of an Activity Queue	188
Reprioritizing a Queued Activity	189
Managing Timer Instances	191
Checking the Status of a Timer	191
Changing the Timer State and Expiration Time	192
Managing Process Attributes	193
Checking the Value and Lock State of a Process Attribute	193
Changing a Process Attribute Value	194
Removing a Process Attribute Lock	195
Checking for Bottlenecks in Process Execution	196
Analyzing Process Execution	197
History Log Schema	198
State Values	198

Monitoring and Managing Two-Phase Commit	199
Managing Two-Phase Commit Operations	202
Chapter 7 Troubleshooting	203
Introduction	203
Process Engine Alarms Window	205
Monitoring Alarms	206
Viewing Alarms	208
Engine Log Files	210
iIS Console Trace Window	212
Messages and Message Filters	212
Specifying Message Filters	213
iIS Message Filters	214
Using the iIS Console Trace Window	215
Setting Message Filters	215
Special Example: Write Client Messages to Trace Window	217
iIS Console Engine Event Filter Window	217
Engine Event Types	218
Using the iIS Console Engine Event Filter Window	218
Displaying All Engine Events	219
Displaying Process Instance Events	219
Filtering Engine Events	221
Performance Charts	221
Viewing Performance Indicators	222
Logging Performance Information	225
Chapter 8 Using the Conductor Script Utility	227
Overview	227
Conductor Script Help	228
Starting Conductor Script	228
Using the Cscript Command	229
Working with Conductor Script	230
General Conductor Script Operations	231
Writing and Executing Scripts	232
Comments	232
Operating System and File Management Commands	233
Managing iIS Process Engines with Conductor Script	235
Starting an Engine	235
Starting Individual Engine Components	236
Example Manual Startup Scenario	238

Managing iIS Process Engines with Conductor Script <i>(continued)</i>	
Monitoring Engines and Engine Components	239
Monitoring the Engine	239
Monitoring Individual Engine Components	240
Changing Engine States	242
Changing Engine Unit States	242
Shutting Down Engine Components	243
Managing Registrations with Conductor Script	243
Making iIS Library Distributions	243
Registering iIS Library Distributions	246
Unregistering iIS Library Distributions	247
Managing Process Execution with Conductor Script	249
Monitoring and Managing Engine Sessions	250
Monitoring and Managing Process Execution	252
Managing Process Instances	252
Managing Activity Instances	252
Managing Activity Queues	253
Managing Timers	254
Managing Process Attributes	254
Checking for Bottlenecks in Process Execution	255
Monitoring and Managing Two-Phase Commit Transactions	258
Monitoring Two-Phase Commit Operations	258
Managing Two-Phase Commit Operations	259
Appendix A Conductor Script Commands	261
Conductor Script Command Summary	261
Environment Mode Commands	263
Engine Mode Commands: Engine Management	264
Engine Mode Commands: Process Execution Management	266
Component Mode Commands	271
Generic component	272
Engine Unit	272
Conductor Script Commands	273
AbortActivity	273
AbortAllProcesses	274
AbortProcess	274
BroadcastMessage	275
CommitTransaction	275
CompleteActivity	276
ConsultActivity	276
CreateActivity	278

Conductor Script Commands *(continued)*

CreateFilter	279
Event Type	279
Object Class	280
Object Instance	280
Operators	280
Examples	281
DelegateActivity	281
DeleteFilter	283
FindDBService	283
FindEngine	284
FindGovernor	284
FindNode	285
FindParentEngine	285
FindPrimary	285
FindUnit	286
FlushLog	286
IIOPServer	287
ListActivities	287
ListActivityQueues	287
ListConductorDistributions	288
ListEngines	289
ListFilters	289
ListProcesses	289
Options	290
ListRegistrations	290
ListSessions	291
ListTimers	291
ListTransactions	291
MakeConductorDistribution	292
ModLogFlags	292
ReadyActivity	293
RegisterAlias	294
RegisterAssignmentRules	294
RegisterProcessDefinition	295
RegisterUserProfile	295
RegisterValidation	296
RemoveReadLock	297

Conductor Script Commands *(continued)*

RemoveWriteLock	297
RollbackActivity	298
RollbackTransaction	298
SendMessage	299
SetAttributeValue	300
SetPassword	300
SetPrimary	301
SetQueuedActivityPriority	301
SetState	302
SetTimer	302
SetTimerDeadline	303
SetTimerElapsed	304
ShowActivity	304
ShowActivityQueue	305
ShowConfiguration	306
ShowEngine	306
ShowLogFlags	307
ShowProcess	307
ShowSession	308
ShowStatus	309
ShowTimer	313
Shutdown	314
StartActivity	314
StartDBService	315
StartEngine	315
StartGovernor	316
StartTimer	317
StartUnit	317
StopTimer	319
SuspendAllSessions	319
SuspendSession	320
TerminateAllSessions	320
TerminateSession	321
Uninstall	321
UnRegisterAlias	322
UnRegisterAssignmentRules	322
UnRegisterProcessDefinition	323
UnRegisterUserProfile	324
WaitForStartup	324

Appendix B Engine Database Schema	327
Database Tables by Category	327
Alphabetical Listing of Tables	329
Database Schema Reference	331
Current State Tables	332
Registration Tables	346
History Log Tables	347
State Values	357
Index	359

List of Figures

Figure 1-1	An iIS Process Management System	30
Figure 1-2	Minimal Engine Configuration	32
Figure 1-3	Engine Configuration with Failover	33
Figure 1-4	Engine Configuration with Both Failover and Load Balancing	35
Figure 1-5	Engine Manager Objects	37
Figure 2-1	Components of an iIS Process System	46
Figure 2-2	iIS Process System Configuration	53
Figure 3-1	iIS Console Main Window	69
Figure 3-2	iIS Console Browser	77
Figure 4-1	Engine Configuration with both Failover and Load Balancing	86
Figure 4-2	Typical Engine Partitioning Scheme	88
Figure 5-1	Registration Steps	140
Figure 5-2	Subprocess Activity References	143
Figure 6-1	Activity State Transitions—from Creation to Termination	164
Figure 6-2	Process Execution Objects: Properties and Relationships	180
Figure 6-3	Client Applications Change Both iIS Process State and Application Data	200
Figure 7-1	Specifying iPlanet UDS Message Output Filters	213
Figure A-1	Hierarchy of Conductor Script Levels	262

List of Procedures

To copy the documentation to a client or server	24
To view and search the documentation	25
Before you invoke the setccomp script, you must do the following	51
To set up an iIS process system	55
To add a new node to an iIS system	61
To create a new iIS Repository Server	62
To move an iIS Repository Server	62
To create a private iIS Repository on a development node	63
To start the iIS Console on Windows or Windows NT	67
To start the iIS Console on UNIX, OpenVMS, or Windows NT	67
To set a password for iIS Console	71
To turn automatic refresh off or on	72
To set the automatic refresh interval	73
To force an immediate refresh	73
To configure a new engine	96
To duplicate an engine configuration	104
To delete an engine configuration	104
To start an engine	110
To start individual engine components	112
To reconfigure an engine	113
To modify database logging for an engine	115
To check the engine runtime configuration	118
To monitor individual engine components	119
To change the state of an engine unit	122
To put the primary engine unit on STANDBY and the backup unit ONLINE	122
To use the Dump/Restore application to dump database tables	127
To preserve custom engine database schema changes when upgrading iIS	132

To use the Dump/Restore application to restore database tables	133
To transfer iIS library distributions to a production environment	141
To register one or more distributions using the iIS Console	146
To unregister a process definition, assignment rule dictionary, or user profile	148
To register an alias using the iIS Console	149
To unregister an alias	150
To view the registrations for a given engine	151
To unregister a registered distribution	151
To monitor instances of a registered process definition	152
To perform a monolithic upgrade	154
To perform a rolling upgrade	155
To obtain state information about a session	174
To suspend a session	176
To terminate a session	177
To send a message to an active session	177
To broadcast a message to all sessions	179
To check the current execution status of a process instance	181
To abort a process instance	184
To change the state of an activity	187
To view the contents of an activity queue	188
To reprioritize a queued activity	189
To change the state and expiration time of a timer	192
To change the value of a process attribute	195
To remove a process attribute lock	195
To find a process bottleneck	196
To register an engine with the alarm service	206
To unregister an engine from the Alarm service	207
To filter alarms in the Process Engine Alarms window	207
To search for an alarm in the Process Engine Alarms window	208
To get detailed information about an alarm	209
To remove an alarm from the Alarm window	209
To view an engine component log file	211
To open the iIS Console Trace window	215
To set message filters for a selected engine	216
To set the engine tracing	217
To open the Engine Event Filter window	218
To view all engine events for the selected engine	219

To view all engine events for an existing process	219
To view all engine events for the next process instance of a specific process definition	220
To specify a custom filter	221
To view performance indicators for an engine	223
To log performance information	225
To stop logging performance information	225
To start Conductor Script on Windows or Windows NT	228
To start Conductor Script on UNIX, OpenVMS, or Windows NT	229
To start an engine	235
To start an engine where delays might be involved	236
To start an individual engine component (governor, database service, or engine unit)	236
To check the engine runtime configuration	239
To monitor individual engine components	240
To change the state of an engine unit	242
To make an iIS library distribution	244
To register an iIS distribution with an engine	246
To unregister an iIS distribution from an engine	248
To obtain state information about a session	250
To suspend or terminate a session	250
To send a message to one or more sessions	250
To check the status of a process instance	252
To abort a process instance	252
To check the status of an activity instance	252
To change the state of an activity	253
To list the contents of an activity queue	253
To reprioritize an activity in a queue	253
To check the status of a timer	254
To change the state of a timer or change its expiration time	254
To check the value or lock state of a process attribute	254
To change the value of a process attribute	255
To remove an attribute lock	255
To find a process bottleneck	255
To resolve transactions after a client or engine failure	260

Preface

The *iIS Process System Guide* describes how to manage a process system, one of the two subsystems that comprise iIS. The guide provides the system management principles and concepts upon which a process system is based, and describes how to perform process system management tasks.

This preface contains the following sections:

- “Product Name Change” on page 19
- “Audience for This Guide” on page 20
- “Organization of This Guide” on page 20
- “Text Conventions” on page 21
- “Other Documentation Resources” on page 22
- “iIS Example Programs” on page 24
- “Viewing and Searching PDF Files” on page 24

Product Name Change

Forte Fusion has been renamed the iPlanet Integration Server. You will see full references to the new name, as well as the abbreviation iIS.

Audience for This Guide

This book assumes familiarity with the basic concepts underlying an iIS process system. A discussion of these concepts and a description of iIS process system components, the functions they perform, and how they interact with an iIS Backbone can be found in the *iIS Conceptual Overview*. Additionally, information about creating and defining business process for use within an iIS process system can be found in the first two chapters of the *iIS Process Development Guide*.

Because an iIS system is built and distributed in an iPlanet UDS environment, it is helpful to understand iPlanet UDS system management concepts. For details, see the *iPlanet UDS System Management Guide*.

Organization of This Guide

The following table briefly describes the contents of each chapter:

Chapter	Description
Chapter 1, "Introduction: iIS Process Management"	Provides a brief introduction to the iIS process engine and process system management.
Chapter 2, "Setting Up an iIS Process Management System"	Describes the components of an iIS process system and how to distribute them among the different nodes in your environment.
Chapter 3, "The iIS Console"	Introduces the iIS Console, an interactive system management tool.
Chapter 4, "Managing Engines"	Explains how to configure, start up, and manage an iIS process engine and its components.
Chapter 5, "Managing Registrations"	Describes the registration process by which process definitions and other development components are made available for execution by a running engine.
Chapter 6, "Managing Process Execution"	Explains how an engine executes process definitions and how to monitor and make adjustments to process execution.
Chapter 7, "Troubleshooting"	Explains tools and techniques used for diagnosing problems in process execution.

Chapter	Description
Chapter 8, “Using the Conductor Script Utility”	Explains how to perform iIS system management tasks using Conductor Script commands.
Appendix A, “Conductor Script Commands”	Provides a command reference for Conductor Script, a command line tool for process system management.
Appendix B, “Engine Database Schema”	Provides a reference for the iIS process engine database schema.

Text Conventions

This section provides information about the conventions used in this document.

Format	Description
<i>italics</i>	Italicized text is used to designate a document title, for emphasis, or for a word or phrase being introduced.
monospace	Monospace text represents example code, commands that you enter on the command line, directory, file, or path names, error message text, class names, method names (including all elements in the signature), package names, reserved words, and URLs.
ALL CAPS	Text in all capitals represents environment variables (FORTE_ROOT) or acronyms (iIS, JSP, iMQ). Uppercase text can also represent a constant. Type uppercase text exactly as shown.
Key+Key	Simultaneous keystrokes are joined with a plus sign: Ctrl+A means press both keys simultaneously.
Key-Key	Consecutive keystrokes are joined with a hyphen: Esc-S means press the Esc key, release it, then press the S key.

Syntax Statements

Syntax statements that describe usage of TOOL methods and script commands use the following conventions:

Format	Description
parentheses ()	Parentheses enclose a parameter list.
comma ,	Commas separate items in a parameter list.
vertical bars	Vertical bars indicate a mutually exclusive choice between items. See braces and brackets, below.
brackets[]	Square brackets to indicate optional values in a syntax statement.
braces { }	Braces indicate a required clause. When a list of items separated by vertical bars is enclosed in braces, you must enter one of the items from the list. Do not enter the braces or vertical bars.
ellipsis ...	The item preceding an ellipsis may be repeated one or more times. When a clause in braces is followed by an ellipsis, you can use the clause one or more times. When a clause in brackets is followed by an ellipsis, you can use the clause zero or more times.

Other Documentation Resources

In addition to this guide, there are additional documentation resources, which are listed in the following sections. The documentation for all iIS products can be found on the iIS CD. Be sure to read [“Viewing and Searching PDF Files” on page 24](#) to learn how to view and search the documentation on the iIS CD.

iIS documentation can also be found online at <http://docs.iplanet.com/docs/manuals/iis.html>.

The titles of the iIS documentation are listed in the following section.

iPlanet Integration Server Documentation

iIS Adapter Development Guide

iIS Backbone Integration Guide

iIS Backbone System Guide

iIS Conceptual Overview

iIS Installation Guide

iIS Process Client Programming Guide

iIS Process Development Guide

iIS Process System Guide

Online Help

When you are using an iIS development application, press the F1 key or use the Help menu to display online help. The help files are also available at the following location in your iIS distribution: `FORTE_ROOT/userapp/forte/cln/*.hlp`.

When you are using a script utility, such as FNscript or Cscript, type help from the script shell for a description of all commands, or help `<command>` for help on a specific command.

Documentation Roadmap

A roadmap to the iIS documentation can be found in the *iIS Conceptual Overview* manual.

iIS Example Programs

iIS example programs are shipped with the iIS product and installed in two locations, one for process development (using the process engine) and one for application integration (using the iIS backbone).

Process Development Examples Process development examples are installed at the following location:

```
FORTE_ROOT/install/examples/conductr
```

The PDF file, `c_examp.pdf`, describes how to install and run the examples in this directory. The Appendix to the *iIS Process Development Guide* also describes how to install and run the examples.

Application Integration Examples Process integration examples are installed at the following location:

```
FORTE_ROOT/install/examples/fusion
```

Each example has its own sub-directory, which contains a README file that explains how to install and run the example.

Viewing and Searching PDF Files

You can view and search iIS documentation PDF files directly from the documentation CD-ROM, store them locally on your computer, or store them on a server for multiuser network access.

NOTE You need Acrobat Reader 4.0+ to view and print the files. Acrobat Reader with Search is recommended and is available as a free download from <http://www.adobe.com>. If you do not use Acrobat Reader with Search, you can only view and print files; you cannot search across the collection of files.

➤ **To copy the documentation to a client or server**

1. Copy the `doc` directory and its contents from the CD-ROM to the client or server hard disk.

You can specify any convenient location for the `doc` directory; the location is not dependent on the iIS distribution. You may want to consolidate your iIS documentation with the documentation for your iPlanet UDS distribution.

2. Set up a directory structure that keeps the `iisdoc.pdf` and the `iis` directory in the same relative location.

The directory structure must be preserved to use the Acrobat search feature.

NOTE To uninstall the documentation, delete the `doc` directory.

➤ **To view and search the documentation**

1. Open the file `iisdoc.pdf`, located in the `doc` directory.
2. Click the Search button at the bottom of the page or select Edit > Search > Query.
3. Enter the word or text string you are looking for in the Find Results Containing Text field of the Adobe Acrobat Search dialog box, and click Search.

A Search Results window displays the documents that contain the desired text. If more than one document from the collection contains the desired text, they are ranked for relevancy.

NOTE For details on how to expand or limit a search query using wild-card characters and operators, see the Adobe Acrobat Help.

4. Click the document title with the highest relevance (usually the first one in the list or with a solid-filled icon) to display the document.

All occurrences of the word or phrase on a page are highlighted.

5. Click the buttons on the Acrobat Reader toolbar or use shortcut keys to navigate through the search results, as shown in the following table:

Toolbar Button	Keyboard Command
Next Highlight	Ctrl+]
Previous Highlight	Ctrl+[
Next Document	Ctrl+Shift+]

To return to the `iisdoc.pdf` file, click the Homepage bookmark at the top of the bookmarks list.

6. To revisit the query results, click the Results button at the bottom of the `iisdoc.pdf` home page or select Edit > Search > Results.

Introduction: iIS Process Management

iIS process system management is largely devoted to managing iIS process engines and the functions they perform.

This chapter provides a high level description of what an iIS process engine is and what it does. It describes the relationship of the engine to other parts of the iIS system and describes its component parts.

This chapter also discusses the tasks required of system managers and the tools they use to perform these tasks.

What Is an iIS Process Engine?

An iIS process engine is the heart of an iIS system, shown in [Figure 1-1 on page 30](#), reproduced from the *iIS Process Development Guide*.

The engine controls and manages business processes from beginning to end, coordinating the work of the different resources that perform the activities defined in each process instance.

At each stage of a process's execution, the engine evaluates whether an activity is ready to be performed and, if so, assigns that activity to the appropriate resources. When an activity is completed, the engine routes the work to the next activity or set of activities.

The engine knows how to manage the flow of a process because it is programmed to control and track that process. The program that the engine executes is an iIS process definition. The process definition is created in the iIS Process Definition Workshop and is then registered with the engine. For information on process definitions, see the *iIS Process Development Guide*.

Once a process definition has been registered with the engine, an iIS process client application (or an application proxy—see [Figure 1-1 on page 30](#)) can open a session with the engine and create an instance of the process. The client application or proxy provides any data required to start the process instance, and then the engine takes over. It assigns activities directly to engine sessions or to queues where they can be accessed. Client applications—or applications interacting with the engine through proxies—perform the work required of each activity using whatever application services, enterprise data, or desktop applications necessary, and then notify the engine that the activity has been completed. For information on client applications, see the *iIS Process Client Programming Guide*. For information about proxies, see the *iIS Backbone System Guide*.

The relationship of the engine to iIS process client applications, on the one hand, and to the process development workshops, on the other, is illustrated in [Figure 1-1 on page 30](#). Client applications maintain sessions with an iIS process engine to initiate a business process or perform its activities in it. The engine can service a very large number of sessions and manage many instances of many different process definitions. As process definitions (as well as assignment rule dictionaries, user profiles, and a validation) are created or modified in the iIS development environment, they can be registered with the engine, and become the basis for further process execution.

The engine is thus central to an iIS system, implementing business processes that require the performance of many activities by many resources.

Multiple Engine Systems

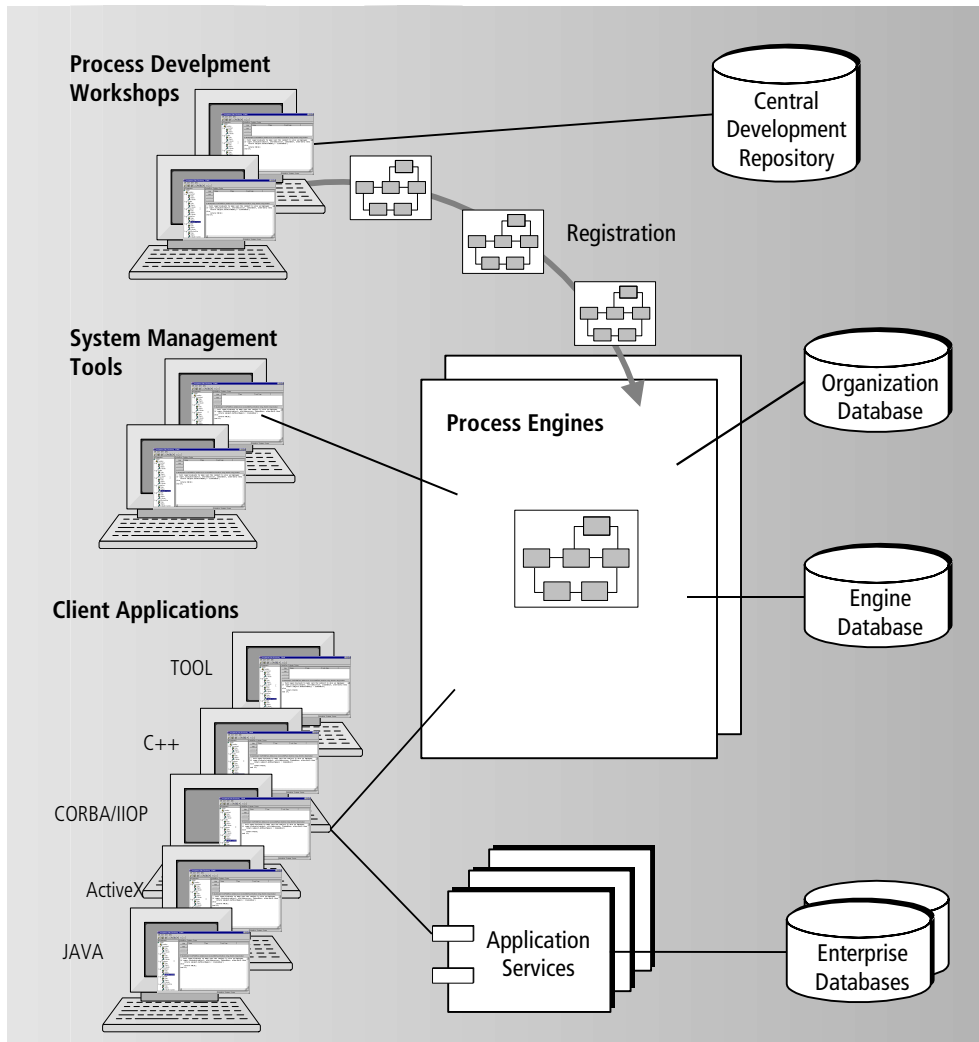
An iIS system does not need to be limited to a single engine. It can have any number of engines. For example, one engine might be used for testing and another used for production. (see [“Production Engines Versus Development Engines” on page 83.](#))

In other cases, organizational considerations or heavy load conditions might require multiple production engines. While engines can communicate with one another—a processes definition that executes on one engine can be invoked as a subprocess by a process definition executing on another—each engine operates independently. A single instance of a process definition executes on only one engine.

In most cases, different process definitions would be executed on different engines. For example, one process might execute on one engine while a subprocess it invokes might execute on another. If the same process definition is executed on more than one engine (to handle the workload of a large number of client users) then client users would normally be divided up between the engines, each user logging in to only one of the engines.

Multiple engines can run in a single iPlanet UDS environment, or can be spread across a number of connected iPlanet UDS environments. However, there is no mechanism by which one engine can fail over to another. iIS uses a different mechanism to handle engine failover (see [“Failover Configuration” on page 33.](#))

Figure 1-1 An iIS Process Management System



iIS Process Engine Components

As mentioned earlier in this chapter, an iIS process engine is the heart of an iIS system and has been designed to support mission-critical production application systems. The iIS process engine maintains performance under heavy and/or increasing loads and provides automatic recovery in case of unanticipated system failure.

To support both failover and load balancing, the iIS process engine is engineered as a set of interacting application components. These applications work together to provide engine failover protection and the flexibility to handle increasing loads. An iIS engine generally consists of the following application components:

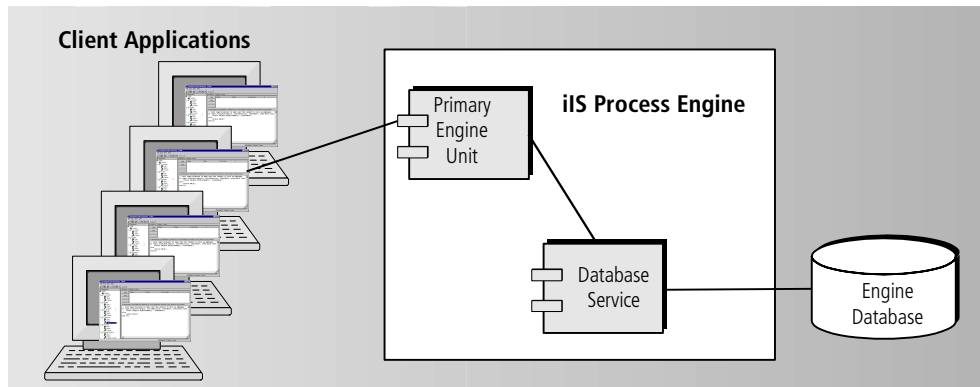
- two engine units
- an engine governor
- a number of database access services

An iIS engine also includes a database that provides storage for persistent state information.

The various engine components are described below in relation to their role in providing failover and load balancing capability. A minimal engine configuration is discussed first, followed by failover and load balancing configurations. To create these configurations, see [“Configuring an Engine” on page 85](#).

Minimal Engine Configuration

An iIS process engine consists minimally of three components—an engine unit, a database service, and a database—as shown in [Figure 1-2](#). In this configuration, the engine provides basic functions without failover or load balancing capabilities.

Figure 1-2 Minimal Engine Configuration

Engine Unit The engine unit is an application that performs all the basic iIS engine functions (see [“What Does an iIS Process Engine Do?”](#) on page 36). The engine unit is a single iPlanet UDS server partition containing a number of manager objects (see [Figure 1-5](#) on page 37). The engine unit, in performing its work, tracks the state of each process, activity, timer, process attribute lock, and session that it creates in the course of process execution. In a typical production situation, the engine unit maintains state information on tens of thousands to millions of objects. It is critical that this state information be kept in a persistent form should the engine unit fail and need to be recovered. For this reason, the engine unit writes all current state information to the engine database (described below).

Database Service The database service is an application that manages the engine unit’s database access. The database service is a single iPlanet UDS server partition that maintains communication channels with the engine unit, opens a session with the engine database, and writes and retrieves state information as required by the engine unit. The database service is a single-threaded application, and can handle only one request to the database at a time. (To see how to overcome this limitation, see [“Full Configuration: Failover and Load Balancing Combined”](#) on page 35.)

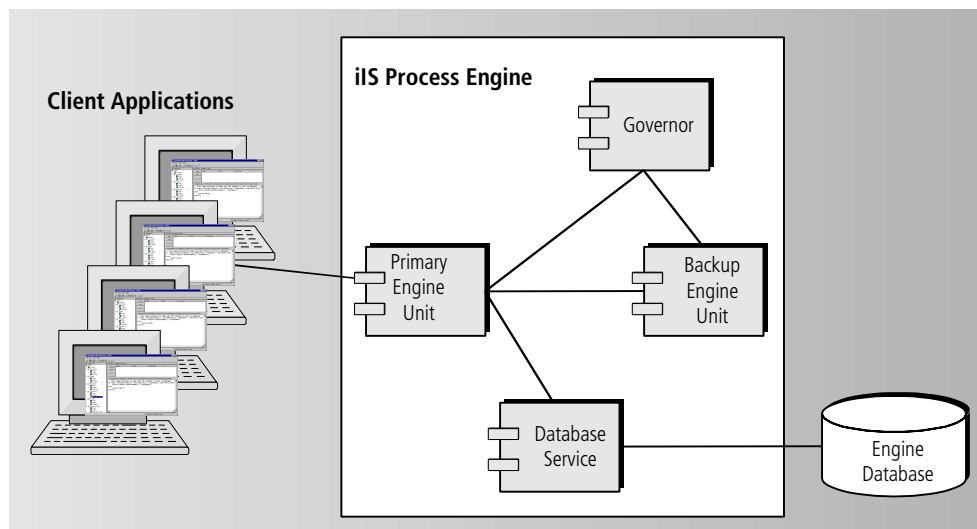
Engine Database The engine database provides persistent storage for state information maintained by the engine. It also stores registration information regarding all process definitions, assignment rule libraries, user profiles, and validations that can be executed by the engine. In addition, the engine database can be used to log historical information for tracking process execution over time. The

engine database is not supplied by the iIS product and must be set up independently; however, the engine unit can create all required database tables at startup time. The iIS process engine supports all relational database systems supported by iPlanet UDS (for example, Sybase, Oracle, Informix, and so on).

Failover Configuration

To provide for failover in case the engine unit fails (for whatever reason), a backup engine unit and governor are added to the iIS process engine configuration, as illustrated in [Figure 1-3](#).

Figure 1-3 Engine Configuration with Failover



Backup Engine Unit The backup engine unit, a replicate of the engine unit partition, has a unique name that distinguishes it from the primary engine unit. The primary and backup engine units are partners. The backup engine unit runs in a standby state when the primary engine unit is on line. The two partners know about each other through an open communication channel. If the primary engine unit fails, the backup engine unit comes on line. In that case, the database service

opens a channel to the backup unit, which takes over by recovering all the persistent state information stored in the engine database. The new primary engine unit starts process execution at the point where the old primary engine unit left it. Client sessions with the engine are maintained throughout this failover transition.

NOTE Failover (and recovery of state information) is not instantaneous. In a production system, the state information stored in the engine database can be quite extensive. It includes information on the state of every process instance, the state of every activity and timer in each process instance, the values of process attributes and their lock states, the state of routers and triggers for each activity instance, session activity lists, queue lists, and so on. Accordingly, the amount of time for failover to occur is dependent on the amount of state information in the database and the speed of database access.

Governor The governor is an application that assists in deciding when failover should occur. The governor maintains an open communication channel with both the primary and backup engine units. It uses these channels to determine if a closing of the communication channel between primary and backup engine units is due to engine unit failure or network failure. For example, if the communication channel between the primary and backup engine units closes, but the governor's communication channel with the primary engine unit remains open, the governor assumes the primary engine unit is still online. However, if neither the governor nor backup engine unit can communicate with the primary engine unit, the governor assumes the primary engine unit has failed, and places the backup engine unit online, making it the new primary engine unit.

(If the primary engine unit has *not* failed, but cannot communicate with either the governor or backup engine unit, the primary engine unit will drop into STANDBY state, since it has no way of knowing the state of its partner.)

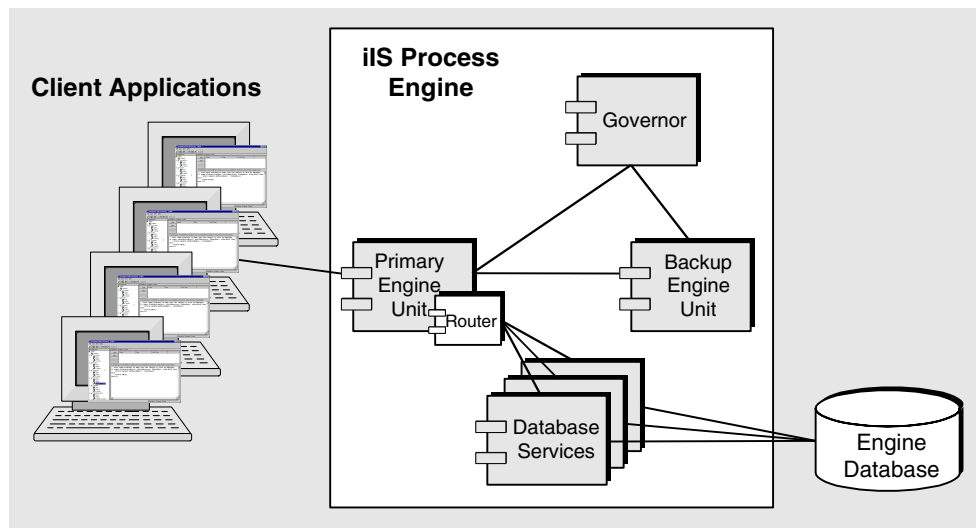
Full Configuration: Failover and Load Balancing Combined

To deal with heavy loads on the engine and overcome the potential bottleneck of a single database service, additional database services can be added to the *failover* engine configuration, as illustrated in [Figure 1-4](#).

With a heavily loaded engine, large numbers of state changes need to be written to the engine database simultaneously. Database access is likely to be a performance problem. To accommodate large loads, database access is shared among a set of database service components (which may be running on different nodes) using a database service router in the primary engine unit.

Each service is assigned a unique name and allocated a priority (depending, perhaps, on its host node—see [“Engine Component Partitioning” on page 85](#)). When the engine unit needs to access the database, the router provides it the highest priority database access service available. If there are a number of services of equal priority available, the router chooses a database service on a round-robin basis, thus sharing the load among all available services.

Figure 1-4 Engine Configuration with Both Failover and Load Balancing



What Does an iIS Process Engine Do?

As suggested in [“What Is an iIS Process Engine?” on page 27](#), an iIS process engine performs a number of different functions in support of an iIS system. These include the following.

Managing sessions The engine opens, suspends, and closes sessions. When opening a session, the engine first validates a user’s logon against an organizational database. For more information on sessions, see [“Engine Sessions” on page 158](#).

Executing processes The engine creates instances of a process and manages their execution from start to finish. During each process’ execution, a succession of activities is performed by applications that have opened sessions with the engine. The engine manages and tracks these activities through to their final completion.

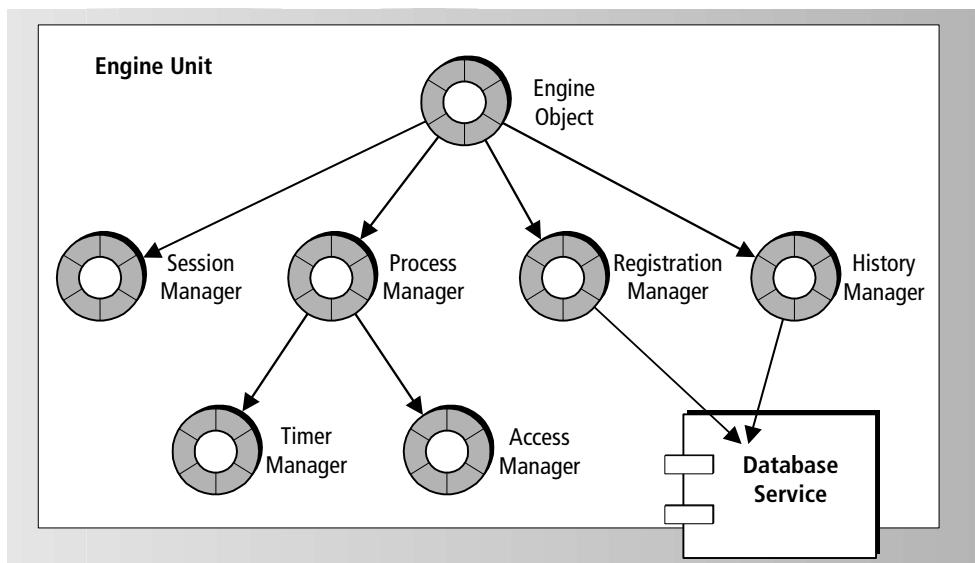
Registering distributions Process definitions, assignment rule dictionaries, user profiles, and a validation constitute the process logic components executed by the engine during process execution. Registration lets you change these components dynamically as business processes and organizational structures change.

Maintaining an engine history database The engine can maintain a history of each process execution, writing all changes of state to an engine history database. The history can include changes of state in activities, process attributes, timers, and so on.

While process execution is the key engine function, managing sessions, registering definitions, and maintaining the history database are all necessary to support process execution.

To provide these functions, the iIS engine is internally structured around a number of manager objects, as shown in [Figure 1-5](#):

- Session Manager—maintains engine sessions
- Process Manager—works closely with the Timer Manager and Access Manger to execute iIS process definitions
- Registration Manager—keeps track of the names and versions of registered process definitions
- History Manager—writes current state and history log information to the engine database.

Figure 1-5 Engine Manager Objects

iIS Process Management Tasks

iIS process management tasks can be grouped into four broad categories:

- setting up and maintaining an iIS process system
- managing iIS process engines
- managing registration of process definitions
- managing process execution

Each of these categories is discussed briefly below and treated in detail in the corresponding chapter of this system management guide.

Setting up and Maintaining an iIS System

Setting up a new iIS process management system—whether used for development, testing, production, or a combination of the three—involves a number of basic tasks, described below.

Setting Up an iPlanet UDS Runtime Environment

An iIS process management system runs in an iPlanet UDS software environment. Both the development and production capabilities of iIS (both process development and process execution) use the distributed runtime services provided by an iPlanet UDS environment. A functioning iPlanet UDS environment is the technical infrastructure underlying an iIS process management system.

Installing iIS Process Management Software

iIS process management system software consists of a number of client and server software modules required to support nodes that perform a number of different functions. The functionally different types of nodes in an iIS process system are the following:

- nodes that support iIS process engine components
- nodes that support process development
- nodes that support client application development or client application execution in a production environment

Before installing iIS process system software, you must first determine which nodes in your iPlanet UDS environment are used for each of these purposes and to install the corresponding system software on each.

When upgrading iIS process system software, especially in production environments, you might have to migrate data from existing iIS process engine databases to new ones. See [“Upgrading an iIS System” on page 63](#).

Setting Up and Maintaining Central Development Repositories

In iIS development environments, the code created in the process development workshops is stored in a modified iPlanet UDS central development repository. As an iIS system manager, you may be called on to create iIS central development repositories and start a Repository Server for each. (Developers can also use private iIS repositories.)

Central development repositories require maintenance, such as backing them up periodically, compacting them regularly, and implementing usage patterns among your development staff that improves performance.

For more information on managing central development repositories, see the *iPlanet UDS System Management Guide*.

Managing iIS Process Engines

Care must be taken to make sure that each iIS process engine is set up and performing properly. Managing engines consists of the following tasks:

- configuring engines
- starting up and shutting down engine components
- monitoring and changing the state of an engine
- reconfiguring an engine to accommodate increased load, improve performance, or recover from failure
- managing engine databases

Managing Registration

An iIS process engine executes process logic created in the process development workshops: process definitions, assignment rule dictionaries, and a user profile. These software components need to be registered with the engine so they can be dynamically loaded by the engine and used in process execution.

As a system manager, you have to regularly register new or updated process definitions (or assignment rule dictionaries or a user profile) with an engine, and possibly unregister them when they become obsolete.

Managing Process Execution

In executing process definitions, the engine creates instances of each process definition and coordinates their execution from start to finish. During process execution, a succession of activities specified by the process definition is performed by client application that have opened sessions with the engine. The engine manages and tracks these activities through to their final completion, ensuring that they are performed in proper sequence.

Managing process execution involves monitoring the activity of the engine and making administrative adjustments when necessary. The specific tasks include:

- monitoring engine sessions, and suspending or terminating sessions when necessary
- monitoring the number of process instances being executed by an engine at any time
- monitoring execution of individual processes, looking for bottlenecks or failures, and aborting or resetting particular process activities, as necessary
- providing historical analysis of process execution activity

iIS Process Management Tools

iIS includes two system management tools: the iIS Console and its command line counterpart, Conductor Script. These tools provide an interface to system management agents in the iPlanet UDS runtime environment and in the iIS process engine for managing engines and process execution.

iPlanet UDS also provides a separate set of command-line tools for managing development repositories.

This section briefly describes the iIS process management tools.

iIS Console

The iIS Console application provides a graphical user interface for performing most of the system management tasks discussed in [“iIS Process Management Tasks” on page 37](#). You use the iIS Console to manage iIS engines (including registering process definitions, and so on), manage individual process instances, and monitor overall process execution activity.

Most system management procedures explained in this guide assume that you are using the iIS Console. For an introduction to the iIS Console, see [Chapter 3, “The iIS Console.”](#)

Conductor Script Utility

The Conductor Script utility is the functional equivalent of the iIS Console application, but with a command-line interface. You can perform any of the functions using Conductor Script commands that you can perform in the iIS Console, but you can also incorporate these functions into scripts for execution at specified times.

For more information on the Conductor Script utility, see [Chapter 8, “Using the Conductor Script Utility.”](#)

Custom System Management Tools

In an iPlanet UDS development environment you can use the iPlanet UDS development workshops to write your own system management tools exercising the same system management agents utilized by the iIS Console and the Conductor Script utility. These custom tools allow you to automate regular system management tasks or respond automatically to particular state conditions in an engine.

Repository Management Tools

iPlanet UDS provides several command-line tools for creating (`rpcreate`), copying (`rpcopy`), starting (`rpstart`), stopping (`rpstop`), and compacting (`rpclean`) central development repositories and Repository Servers.

For more information on repository management tools, see the *iPlanet UDS System Management Guide*.

Dump/Restore Facilities

iIS provides facilities for dumping the information in an engine database to an ASCII file. These facilities—a Dump/Restore graphical user interface application and an equivalent command-line utility—are used to restore an engine database in iIS upgrades in which the engine database schema is changed. For more information on the Dump/Restore utilities, see [“Dumping and Restoring Data” on page 126](#).

Setting Up an iIS Process Management System

Setting up an iIS process management system—whether for development, testing, production, or a combination of the three—is the first task you must perform as an iIS process system manager.

System setup involves determining which nodes in your environment support the different components of your iIS process system, and then installing the appropriate iIS software on each of these nodes.

Since iIS runs on top of a functioning iPlanet UDS environment, you may have to set up an iPlanet UDS environment before setting up your iIS process system.

This chapter explains how to set up an iIS process system, and covers the following topics:

- setting up an iPlanet UDS runtime environment
- components of an iIS process system
- setting up an iIS process system
- maintaining an iIS process system

For information on how to configure and start an iIS process engine, refer to [Chapter 4, “Managing Engines.”](#)

Preparation: Setting up an iPlanet UDS Runtime Environment

An iIS process system runs in an iPlanet UDS software environment. Both the development and runtime capabilities of iIS (process development and process execution) use the distributed runtime services provided by an iPlanet UDS environment. A functioning iPlanet UDS runtime system (including a central development repository for iIS process development) is the technical infrastructure underlying an iIS process system.

Consequently, you must have a functioning iPlanet UDS environment before setting up an iIS process system. This section introduces some of the important considerations for setting up an iPlanet UDS environment. For detailed information on how to set up an iPlanet UDS environment, consult the *iPlanet UDS System Management Guide*.

iPlanet UDS system software is designed to be environment independent. Nevertheless, iPlanet UDS software has critical dependencies on the specific versions of operating systems, window systems, networking systems, runtime libraries, and database management systems used at your site. One of the most important aspects of properly setting up an iPlanet UDS environment is to make sure that the physical environment (hardware and software) meets iPlanet UDS system requirements. For more information on these requirements, see the *iPlanet UDS System Installation Guide*.

While iPlanet UDS system software is usually installed in an existing network configuration, you should consider how to best use the available resources, depending on the type of environment you are setting up.

If the environment is principally an iPlanet UDS or iIS *development* environment, then the locations of your central development repositories (Repository Servers) and Environment Manager service are key considerations.

If the environment is principally a *production* environment for mission-critical applications, then the availability and locations of vital application resources (such as database management systems and C program libraries) and the speed and reliability of servers are some key considerations. These considerations might also influence where you place the Environment Manager service.

In general, when setting up an iPlanet UDS environment, you first install iPlanet UDS on a node that plays the role of a central server. Typically, this node hosts the Environment Manager service and the Repository Server and often serves as a central distribution node for installing iIS on other nodes in your environment. After you install iPlanet UDS on your central server, you install iPlanet UDS on other nodes in the environment.

The iPlanet UDS installation program creates an iPlanet UDS directory structure and source files on your target node, sets a number of environment variables used by iPlanet UDS at startup time, and creates startup scripts for starting the appropriate system management service—the Environment Manager or Node Manager—for your target node.

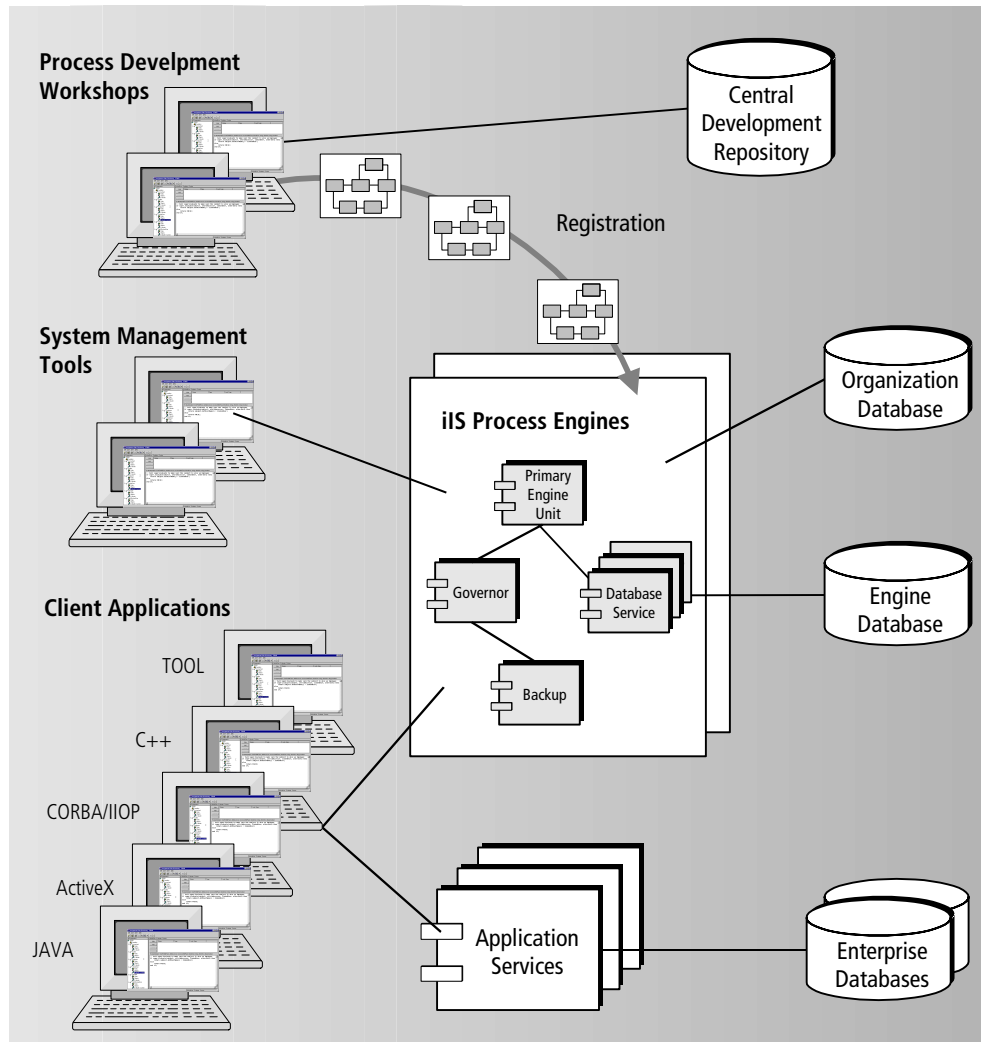
Setting up an iIS process system, in turn, depends upon the iPlanet UDS system management services provided in an iPlanet UDS environment.

iIS Process System Components

An iIS process system consists of both development and runtime components, as illustrated in [Figure 2-1](#), and described in the *iIS Process Development Guide*.

The system includes one or more iIS process engines, which are accessed by a number of iIS process client applications. An engine controls and manages the flow of business processes, coordinating the work of the different client applications that perform the activities that make up these processes. The client applications perform activities directly or by invoking other application services in a distributed application.

An iIS process engine executes process definitions created by developers using a set of graphically-based process development workshops. These workshops store information in a central development repository. When development of a process definition is complete, the process definition is dynamically registered with an engine, which can then execute the process on behalf of a number of applications. These applications can be client applications that directly call the engine, or applications that are integrated using the iIS backbone.

Figure 2-1 Components of an iIS Process System

The components of an iIS process system, shown in **Figure 2-1**, are implemented through a number of software modules running on various nodes in your computing environment. Some of the components are provided by iIS system software (iIS process engine components, process development workshops, and system management tools). Some must be developed on site and require iIS system software (process definitions and iIS process client applications). Others must be developed on-site using other software products (organization database).

To set up an iIS process system, install the iIS system software modules that support development on your development nodes. Then install the iIS runtime modules on the nodes on which you will be executing your distributed enterprise applications.

iIS Backbone

If you are integrating applications with the iIS Backbone, also install the iIS runtime. For more information, refer to the *iIS Installation Guide*. For management functions specific to the iIS Backbone, refer to the *iIS Backbone System Guide*.

iIS Process System Software

iIS process system software consists of a number of software modules that provide—or are used to develop and manage—many of the iIS process system components illustrated in [Figure 2-1 on page 46](#). These modules, organized by functional category, are described in the following table. The table also indicates which system software modules are marked for use as compiled partitions.

Table 2-1 Process System Software Modules

Software Module	Category	Description
WFEEngineUnit (compiled)	Engine component	The server application that performs all the basic iIS engine functions (see “What Does an iIS Process Engine Do?” on page 36). Consists of a single iPlanet UDS server partition. Needed on engine server nodes.
WFGovernor (compiled)	Engine component	The server application that assists in deciding when failover of an engine unit should occur (see “Governor” on page 105). Consists of a single iPlanet UDS server partition. Needed on engine server nodes.
WFDatabaseSvc (compiled)	Engine component	The server application that manages the engine unit’s database access (see “Database Service” on page 109). Consists of a single iPlanet UDS server partition. Needed on engine server nodes.

Table 2-1 Process System Software Modules (*Continued*)

Software Module	Category	Description
WFLibrary (compiled)	Process development	A library distribution that supports the process development workshops. Needed in iIS process development repository and on process development nodes.
WFModel (compiled)	Process development	A library distribution that provides repository schema definitions used to create code in the process development workshops. Needed in iIS process development repository and on process development nodes.
WFAccessServiceObj (compiled)	Process development	A library distribution that provides code used in the process development workshops for accessing iPlanet UDS service objects from within activity and router methods. Needed in iIS process development repository and on engine server nodes.
WFClientLibrary (compiled)	TOOL client application development	A library distribution that provides the API for process client applications written in the iPlanet UDS TOOL language. Needed in client application development repository, on TOOL client application development nodes, and on runtime nodes running client applications.
WFProcMgrLibrary	TOOL client application development	A library distribution that provides the limited process management API for process client applications written in the iPlanet UDS TOOL language. Optionally used in client application development repository, on TOOL client application development nodes, and on runtime nodes running such client applications.
WFAdminLibrary	TOOL client application development	A library distribution that provides the limited process execution monitoring API for process client applications written in the iPlanet UDS TOOL language. Optionally used in client application development repository, on TOOL client application development nodes, and on runtime nodes running such client applications.

Table 2-1 Process System Software Modules (*Continued*)

Software Module	Category	Description
WFClientAPI (compiled)	C++ client application development	A DLL required to write process client application code in C++. Needed on C++ client application development nodes, and on runtime nodes running C++ client applications.
WFCORBAapi (compiled)	Java/IIOP client application development	A server application (CORBA interface service) required to execute process client application code written in JAVA/IIOP. Needed on at least one server node in the iIS environment, normally the central server or an engine server.
WFConsole	System Management	A GUI client application for performing iIS process management system tasks. For details, see Chapter 3, "The iIS Console."
WFScript	System Management	A command-line client application for scripting iIS system management tasks. For details, see Chapter 8, "Using the Conductor Script Utility."
WFDrWind	Engine Database Dump/Restore application	A graphical user interface application used for dumping engine database tables to a file and for restoring data to an engine database. For details, see "Dumping and Restoring Data" on page 126.
WFDrDump	Engine Database Dump utility	A standalone program that dumps the engine state and history data to files. For details, see "Dumping Database Tables" on page 126.
WFDrRest	Engine Database Restore utility	A standalone program that restores the data from files generated using WFDrDump to an engine database. For details, see "Restoring Database Tables" on page 131.
OFCustomIF (compiled)	System	A library distribution that provides "interface glue" between a client application, engine, and user profile. Needed in iIS development repositories, on engine server nodes, and on process client application development nodes.
WFCustomIF (compiled)	System	Extension of OFCustomIF.

Table 2-1 Process System Software Modules (*Continued*)

Software Module	Category	Description
WFEEnvDefinition (compiled)	System	A library distribution that provides underlying system management support for accessing iIS process engines and performing registration of process definitions. Needed on all nodes.
WFEEnvAgent (compiled)	System	An application that supports generation and deployment of library distributions for registration of process definitions, assignment rule dictionaries, and user profiles. Needed on central server node.
en_us.cat	System	A message catalog used in translating all language strings in the product.
WFSeed.btx WFSeed.btd	Development workshops	A base development repository that contains all the class definitions needed to create process definitions or TOOL client applications. Needed on repository server nodes.

Compiled and Interpreted Engine Components

There is a script for alternating between the use of compiled or interpreted engine components. This is useful if exceptions occur when you are running an engine in compiled mode; you can switch to interpreted mode to obtain trace back on exceptions. The platform-specific script, which is installed in FORTE_ROOT/install/bin, is the following:

Script Name	Platforms	Types of Script
setccomp.sh	All UNIX	Bourne-shell
setccomp.bat	MS Windows NT	MS-DOS batch file
setccomp.com	Alpha VMS, VAX/VMS	VMS command procedure

Syntax:

```
setccomp.sh TRUE | FALSE
```

TRUE specifies compiled partitions; FALSE specifies interpreted partitions.

- **Before you invoke the setccomp script, you must do the following**
1. Set the FORTE_ROOT environment variable to point to a valid iIS installation.
 2. Install the following iIS library distributions in the iPlanet UDS development environment:
 - WFGovernor
 - WFEngineUnit
 - WFDatabaseSvc
 3. Start the iPlanet UDS Environment Manager on the server node where you invoke the script.

iIS Process System Configuration

Setting up an iIS process system involves a bit of system design work. You must decide in advance, for example, which nodes in your computing environment host the various iIS process system components shown in [Figure 2-1 on page 46](#).

In particular, you must decide which node or nodes in your environment support the following functions:

- running iIS process engine components
- developing process definitions
- developing iIS process client applications
- hosting iIS system management tools
- running the iIS central development Repository Server
- running iIS process client applications

All but the last of these functions require iIS process system software. In setting up an iIS process management system, these are the components that are the focus of attention.

For purposes of discussion, an example iIS process system configuration is shown in [Figure 2-2 on page 53](#). Seven distinct kinds of iIS nodes are shown:

Central server The iPlanet UDS central server node in your environment. It hosts the iPlanet UDS Environment Manager (including Name Service) and one or more unique iIS services (such as the WFEnvAgent service).

Repository server A server node (runs an iPlanet UDS Node Manager) that hosts the iIS central development repository and repository service.

Engine server A server node (runs an iPlanet UDS Node Manager) that hosts one or more engine components of one or more engines. One of the engine servers typically hosts an engine database. (Putting the engine database and the engine server on separate nodes is generally not recommended for performance reasons.)

Process development A node on which process developers (using the process development workshops) create the process definitions that are executed by iIS process engines.

Client application development A node on which client application developers create the process client applications that communicate with the iIS engine and perform the activities specified in process definitions.

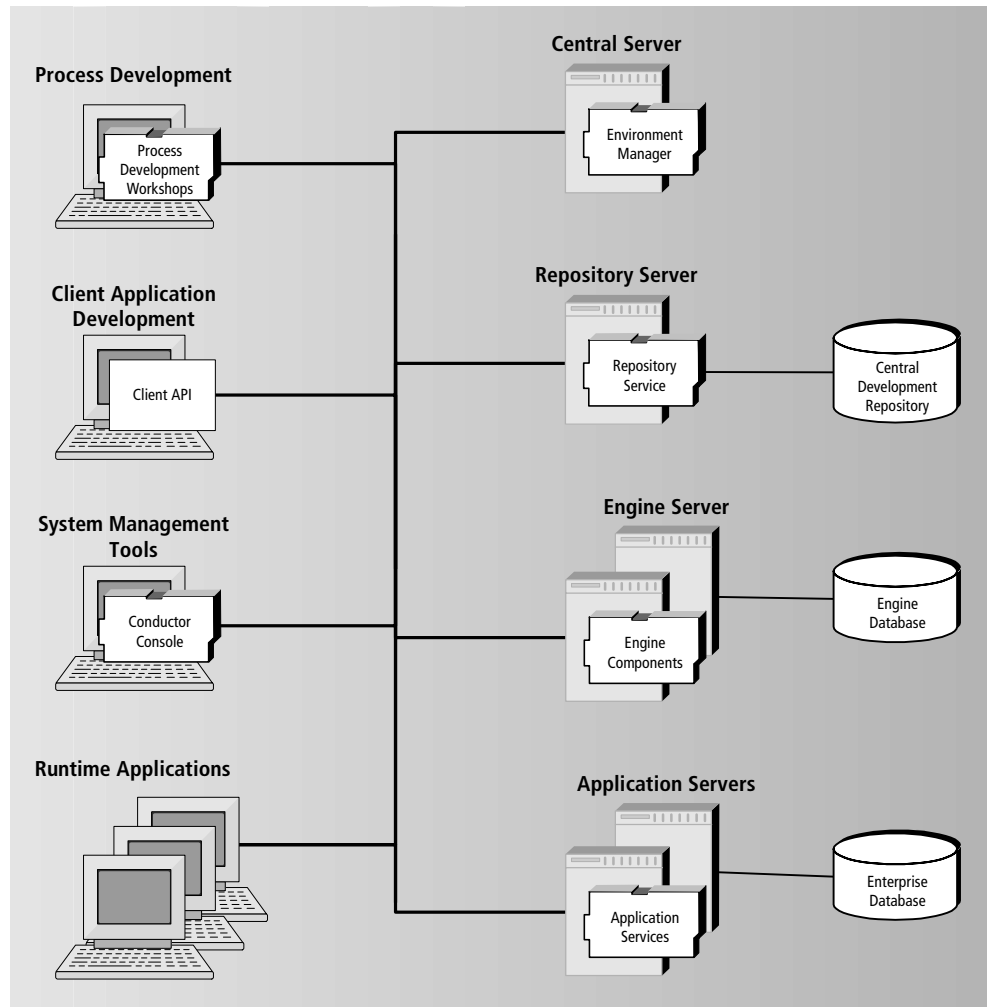
System management A node used by a system manager to manage an iIS process system.

Runtime client A node that runs an iIS process client application, but which cannot be used for development.

While conceptually it is easier to think of each of these nodes as distinct, in reality there is no reason why some of the functions described could not be combined and performed on a single node. (In fact, you could run the whole system on a single node.)

For example, process development and client application development could both be performed on the same client node, or each could be performed on an engine server node that supports a windowing system. Deciding where these components are installed depends principally on who performs the various iIS development and system management tasks.

Also, the central server, which plays a unique role in an iIS system, can both host the central development repository and serve as an engine server node, supporting one or more iIS process engine components. Application servers in your iIS process system can also host iIS engine components. Choosing which components to install on which servers depends on the resources available on each server and the requirements of each component.

Figure 2-2 iIS Process System Configuration

You do not have to decide at iIS installation time exactly which engine components to assign to each server node because all component partitions are installed on each server node that supports an engine. You select which components to start on any node when you configure each engine (see [“Configuring an Engine”](#) on page 85).

For the purpose of system setup, the primary considerations are that the iIS process engine is the workhorse of your iIS process system and that it accesses a potentially very large database. Your engine server node or nodes should therefore be a very high performance server with plenty of available memory. If you are unsure at this point how to configure your iIS engine, designate all likely nodes as engine server nodes. This approach gives you the most flexibility later on.

Another design issue concerns the iIS development repository. You might want iIS developers to use an existing iPlanet UDS repository, or you might prefer to create one or more central development repositories for exclusive use by iIS developers. In addition, you might decide that the repository server needed to support iIS development should reside on a server other than the central server node hosting the Environment Manager, as shown in [Figure 2-1 on page 46](#).

In any case, before beginning your iIS setup procedure, fill in a table similar to the one that follows:

iIS Function	Node type: Server/Client	Node Names
Central server	Server	
Repository server	Server	
Engine servers	Server	
Process development	Client or server	
Client application development	Client or server	
System management	Client or server	
Runtime client application	Client or server	

Setting Up an iIS Process System

After you have decided how to configure your iIS process system (and filled out the table in the previous section), you are ready to set up your system.

To set up an iIS process system, run the platform-specific iIS installation program on each node requiring iIS process system software. The software components installed by the program depend on whether the target node is a central server node, engine server node, client development node (process development or client application development), or runtime client.

Because the iIS installation program depends on services provided by an iPlanet UDS runtime system, you must set up a functioning iPlanet UDS runtime environment if you do not already have one at your site. (See [“Preparation: Setting up an iPlanet UDS Runtime Environment”](#) on page 44.)

iIS Process System Setup Procedure

The following general procedure should be followed in setting up an iIS process system.

► To set up an iIS process system

1. Make sure that you have a functioning iPlanet UDS environment that includes all the server and client nodes required to support your iIS process system.
2. Run the iIS installation program on your iPlanet UDS central server node.
3. Run the iIS installation program on all iIS engine server nodes.
4. Run the iIS installation program on all iIS client nodes (process development and/or client application development nodes).
5. On all development nodes, set the FORTE_REPOSNAME environment variable to the name of your iIS Repository Server. (You can use the iPlanet UDS Control Panel—see the *iPlanet UDS System Management Guide* for more information.)
6. Configure and start your iIS process engine or engines. (See [Chapter 4, “Managing Engines.”](#))

7. After process client applications have been developed, run the iIS installation program on all runtime client nodes (those nodes that support an iIS process client application but do not support development).
8. In sites developing or running process client applications using the CORBA/IIOP interface to the iIS engine, start the CORBA/IIOP interface service using the IIOPServer Conductor Script command (see “IIOPServer” on page 287).

This setup procedure creates a functioning iIS process system in which developers can create process definitions and register them with an iIS engine and then develop process client applications and use them to perform the activities of an executing process instance.

CAUTION An iIS process system can only function in the iPlanet UDS environment in which it has been set up. For example, you cannot change the name of your iPlanet UDS environment—even if the physical environment remains the same—and still have a functioning iIS process system.

The iIS Installation Program

The setup procedure described in the previous section makes extensive use of the iIS installation program. The iIS installation program has four main installation options: central server node, engine server node, development client node, and runtime client node. Each installs, by default, the iIS system software appropriate for that type of node, and provides customizing options (such as installing iIS process management tools), as well.

This section describes what the installation program does on each of the following four node types:

- central server
- engine server
- client development (process development or client application development)
- runtime client

For platform-specific information and installation program details, see the *iIS Installation Guide*.

Central Server Installation

On a central server node—the first node to be installed in an iIS system—the iIS installation program performs the operations described in the following table:

Operation	Software Module	FORTE_ROOT directory
Prepares node as the central distribution node for all distributions.	All application distributions All library distributions	appdist
Loads distributions into environment repository to enable deployment by iPlanet UDS system management agents.	All server application distributions All library distributions	—
Transfers iIS repository files to node.	WFSeed.btx WFSeed.btd	install/respos cpy
Creates iIS repository server.		repos
Installs engine components.	WFEngineUnit WFGovernor WFDatabaseSvc	userapp
Installs process development libraries.	WFLibrary WFModel WFAccessServiceObj	userapp
Installs iIS TOOL client development libraries.	WFClientLibrary WFProcMgrLibrary WFAdminLibrary	userapp
Installs iIS C++ client development DLL.	WFClientAPI	userapp
Installs CORBA/IIOP interface service.	WFCORBAapi	userapp
Installs iIS process management client applications.	WFConsole WFScript	userapp
Installs iIS system libraries.	OFCustomIF WFCustomIF WFEEnvDefinition	userapp
Installs registration support service.	WFEEnvAgent	userapp
Installs message catalog file	en_us.cat	install/nls/co nductr
Installs Dump/Restore facilities	WFDrWind WFDrDump WFDrRest	userapp

Engine Server Installation

On an engine server node, which must be installed after the central server node, the iIS installation program performs the operations described in the following table:

Operation	Software Module	FORTE_ROOT directory
Transfers iIS repository files to node.	WFSeed.btx WFSeed.btd	install/resposcpy
Installs engine components.	WFEUnit WFGovernor WFDatabaseSvc	userapp
Installs process development libraries.	WFLibrary WFModel WFAccessServiceObj	userapp
Installs iIS TOOL client development libraries.	WFClientLibrary WFProcMgrLibrary WFAdminLibrary	userapp
Installs iIS C++ client development DLL.	WFClientAPI	userapp
Installs iIS system management client applications.	WFConsole WFScript	userapp
Installs iIS system libraries.	OFCustomIF WFCustomIF WFEEnvDefinition	userapp
Installs message catalog file	en_us.cat	install/nls/conductr
Installs Dump/Restore facilities	WFDWind WFDWindDump WFDWindRest	userapp

Development Client Installation

A development client node can be used for either process development or client application development. On a development client node, which must be installed after the central server node, the iIS installation program performs the operations described in the following table:

Operation	Software Module	FORTE_ROOT directory
Transfers iIS repository files to node.	WFSeed.btx WFSeed.btd	install/resposcpy
Installs process development libraries.	WFLibrary WFModel WFAccessServiceObj	userapp
Installs iIS TOOL client development libraries.	WFClientLibrary WFProcMgrLibrary WFAdminLibrary	userapp
Installs iIS C++ client development DLL.	WFClientAPI	userapp
Installs iIS process management client applications.	WFConsole WFScript	userapp
Installs iIS system libraries.	OFCustomIF WFCustomIF WFEnvDefinition	userapp
Installs message catalog file	en_us.cat	install/nls/conductr

Runtime Client Installation

On a runtime client node, which must be installed after the central server node, the iIS installation program performs the operations described in the following table:

Operation	Software Module	FORTE_ROOT directory
Installs iIS TOOL client libraries.	WFClientLibrary WFProcMgrLibrary WFAdminLibrary	userapp
Installs iIS C++ client development DLL.	WFClientAPI	userapp
Installs iIS process management client applications.	WFConsole WFScript	userapp
Installs iIS system libraries.	OFCustomIF WFCustomIF WFEnvDefinition	userapp
Installs message catalog file	en_us.cat	install/nls/conductr

Configuring and Starting Your iIS Engines

When your iIS process system software is installed on the various nodes in your environment (including an iIS central development repository), an iIS application system designer can begin implementing the various design elements, process developers can begin creating process definitions, and client application developers can begin writing iIS client applications.

However, before long it will be necessary to have a test engine running in your environment. Configuring and starting an iIS process engine requires familiarity with iIS process management tools (see [Chapter 3, "The iIS Console"](#)) and is described fully in [Chapter 4, "Managing Engines."](#)

Maintaining an iIS Process System

Once your iIS process system is set up and functional (including configuration and startup of engines, as discussed in [Chapter 4, “Managing Engines”](#)), there are relatively few maintenance tasks that you need to perform.

Most maintenance concerns the databases in the system. For example, you need to regularly back up the central development repository as well as compact it from time to time (see the *iPlanet UDS System Management Guide* for more information). Also the engine database must be monitored to make sure you do not run out of disk space.

Other maintenance tasks might include the following:

- adding new nodes to an iIS system
- moving an iIS process engine
- creating a new iIS Repository Server
- moving an iIS Repository Server
- creating a private iIS repository
- uninstalling an iIS system

Each of these are discussed briefly below.

Adding New Nodes to an iIS System

► To add a new node to an iIS system

1. Decide which function the new node is to perform in your iIS system.
2. Run the iIS installation program and choose the appropriate installation option.

3. Choose default or custom installation, depending on the components you want to install on the node.

NOTE If you know the application or library distributions required for the new node (see [Table 2-1 on page 47](#)), you can install them using iPlanet UDS system management services rather than the iIS installation program. The distributions can be installed on the new node using Escript or the iPlanet UDS Environment Console, depending on whether the new node is a client or server, as described in the *iPlanet UDS System Management Guide*.

Moving an iIS Engine

Moving an iIS engine or engine component is straightforward. You move the engine or engine component to a different engine server node by reconfiguring the engine as described in [“Configuring an Engine” on page 85](#). If the target node is not presently an engine server node, install iIS system software on the target node using the instructions in [“Adding New Nodes to an iIS System” on page 61](#).

Creating a New iIS Repository Server

- **To create a new iIS Repository Server**
 1. Decide which node is to host the new repository server.
 2. Run the iIS installation program and choose the Engine Server option.
 3. Choose Custom Installation.
 4. Check Create iIS Repository and clear all other options.
 5. Click OK and follow the screen questions and instructions.

Moving an iIS Repository Server

- **To move an iIS Repository Server**
 1. Select a server node to host the relocated repository server.
 2. Shut down the current Repository Server.

3. Move the two iIS repository files (*reposname.btx* and *reposname.btd*) from FORTE_ROOT/repos on the original Repository Server node to the same location on the target node.
4. Modify the iPlanet UDS startup script on the server to include the `rpstart` command needed to start the iIS repository server (see the *iPlanet UDS System Management Guide*).

Creating a Private iIS Repository

► To create a private iIS Repository on a development node

1. Copy the two repository seed files (*wfseed.btx* and *wfseed.btd*) from FORTE_ROOT/resposcpy to FORTE_ROOT/repos.
2. Rename the two files to correspond to the name of your iIS private repository. You might have to modify the files to provide read/write access.
3. Edit the iPlanet UDS Distributed or iPlanet UDS Standalone system icons that start iIS.

Set the `-fr` flag in the icon's command line to reference the new private repository. For more information, refer to instructions about starting the process development environment in the *iIS Process Development Guide*.

Upgrading an iIS System

Upgrading an iIS system is generally straightforward: You install the new version of iIS system software over the older version. Engine configuration files can be used with the new version, though you might want to reconfigure engines to make use of new engine configuration options.

The situation in which you must exercise caution, however, is if you have state or history information in your engine database that you need to preserve across the upgrade. If the engine database schema changes from the older version to the newer version of iIS, you must migrate that data from the older database schema to the new one. iIS provides Dump/Restore facilities to use for this purpose. See [“Dumping and Restoring Data” on page 126](#), and be sure to check all release notes.

Uninstalling an iIS System

Because iIS server software is installed using iPlanet UDS system management services, the iPlanet UDS environment repository has a record of all engine components installed on servers in your environment. To uninstall an iIS system, first use iPlanet UDS system management tools (Environment Console or Escript) to uninstall these components from the environment.

In addition, uninstalling an iIS system requires removing the various iIS system software components from the various nodes on which they have been installed. The components—application partitions and libraries—are installed in a standard iPlanet UDS location on each node: `FORTE_ROOT/userapp/distribution_ID`, where `distribution_ID` is the name of the software module. (For the names of the software modules installed on each type of node, see [“The iIS Installation Program”](#) on page 56.)

To revert a repository to a non-iIS development repository, you need to delete all the libraries imported by the iIS installer. After deleting the WFLibrary distribution from `FORTE_ROOT/userapp`, the iIS-specific buttons and menus are removed from the Repository Workshop, and no longer appear on startup.

The iIS Console

This chapter describes the iIS Console, the iIS product's window-based tool for performing iIS process management tasks.

This chapter assumes that you have set up an iIS process management system and have installed iIS process management applications (iIS Console and Conductor Script) on at least one node in your environment.

This chapter covers the following topics:

- overview of the iIS Console
- starting the iIS Console
- using the iIS Console window
- navigating to other iIS Console windows
- summary of menu commands

Overview

The iIS Console is the main system management application for performing iIS process management tasks. The iIS Console has a graphical user interface, as opposed to Conductor Script, which has a command line interface.

NOTE Neither the iIS Console or Conductor Script can be used to perform backbone system management tasks. For information on backbone system management, refer to the *iIS Backbone System Guide*.

The iIS Console is used for the following four general kinds of iIS process management tasks:

- configure and manage process engines

The iIS Console provides a set of windows that let you configure a new iIS engine, start and view the status of an already configured engine, and reconfigure an existing engine, if necessary.

- manage registration

You can register a number of process logic components created in the iIS process development workshops with one or more iIS engines. When these software components are registered, they can be dynamically loaded by an engine and used in process execution.

- manage process execution

Using the iIS Console, you can monitor the activity of an engine and make administrative adjustments when necessary. You can monitor engine sessions and send messages to client applications, and you can monitor the engine's execution of individual processes.

- monitor alarms and diagnostic messages

You can monitor high priority events (alarms) from within the iIS Console, as well as view diagnostic messages generated by engine components. All these alarms and messages are useful in troubleshooting problems in engine performance and process execution.

The iIS Console connects to and communicates with the executing iPlanet UDS Environment Manager and any active Node Managers in your iPlanet UDS environment. It gives you access to iIS engine agents: objects that control engine components and provide information about process execution.

NOTE The Conductor Script command line utility is functionally equivalent to the iIS Console. However, all procedural instructions in this guide assume you are using the iIS Console. For information on Conductor Script, see [Appendix A, "Conductor Script Commands."](#)

Starting the iIS Console

You can start the iIS Console on any node in your iPlanet UDS environment where it has been installed.

- **To start the iIS Console on Windows or Windows NT**
 1. Double-click the iIS Console icon.
 2. Enter a valid password if one is requested (see [“Setting Password Protection for iIS Console” on page 71](#)).
- **To start the iIS Console on UNIX, OpenVMS, or Windows NT**
 1. Use the `cconsole` command (see [“Using the cconsole Command”](#) below for information).
 2. Enter a valid password if one is requested (see [“Setting Password Protection for iIS Console” on page 71](#)).

When the iIS Console starts, it opens the iIS Console main window shown in [Figure 3-1 on page 69](#).

Using the `cconsole` Command

As mentioned above, you start the iIS Console on command-line-based operating systems by executing the `cconsole` command.

The syntax of the `cconsole` command for most platforms:

```
cconsole [-fl log_flags] [-fm memory_flags]
```

The syntax of the `cconsole` command for OpenVMS:

```
VFORTE CCONSOLE
  [/LOGGER=log_flags]
  [/MEMORY=memory_flags]
```

As in all iPlanet UDS command line specifications, if you use a name that includes a space, you should enclose the name in double quotation marks.

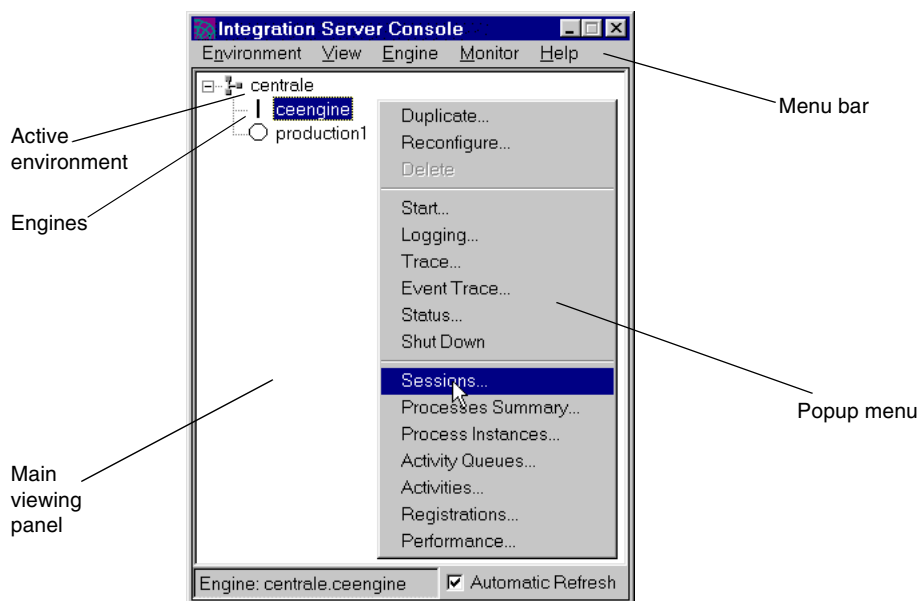
The following table describes the command line flags for the `cconsole` command.

Flag	Description
<code>-fl log_flags</code> <code>/LOGGER=log_flags</code>	Specifies the log flags to use for the iIS Console session. See the <i>iPlanet UDS System Management Guide</i> for information about the syntax for specifying log flags. Overrides the <code>FORTE_LOGGER_SETUP</code> environment variable setting. On UNIX, you must specify the log flags in double quotes.
<code>-fm memory_flags</code> <code>/MEMORY=</code> <code>memory_flags</code>	Specifies the memory flags to use for the iIS Console session. See the <i>iPlanet UDS System Management Guide</i> for syntax information. Overrides defaults appropriate for the operating system. On UNIX, you must specify the memory flags in double quotes.

The iIS Console Main Window





The iIS Console main window is a view of configured iIS process engines in your active environment. In the main window, you can initiate all system management tasks you want to perform in your iIS process system. You can use the main window to manage engines, registrations, and process execution in your environment. You can also add password protected access to iIS Console.

The iIS Console main window consists of two areas: the main viewing panel and the menu bar.

Figure 3-1 iIS Console Main Window

Main Viewing Panel

The main viewing panel displays all configured engines in an iPlanet UDS environment. Engines are shown with icons representing the state of each engine:

	Off	Engine is configured but has not been started.
	Transition	Engine is in the process of starting up or shutting down.
	Online	Engine has fully started and is online.
	Error state	Engine has encountered an exception while attempting to start.

Most tasks you perform using the iIS Console are initiated by first selecting an engine from the main viewing panel and then selecting the desired operation from the appropriate menu.

Menu Bar

The iIS Console menu bar provides all the commands you can execute from the main window. The menus are summarized below, and a full list of the commands is provided at the end of this chapter.

The main window menus are:

Environment menu Provides environment-wide iIS Console commands (such as for registrations), or general commands such as opening a window to see alerts being generated from iIS process engines.

View menu Provides commands for setting refresh properties of the window.

Engine menu Provides engine management commands for configuring, starting up, viewing the status of, deleting, performing logging and trace functions, and shutting down an iIS engine.

Monitor menu Provides commands for monitoring and managing processes being executed by an iIS engine, verifying registrations, and charting performance.

Help menu Provides online Help for the iIS Console.

Mouse Popup Menu

iIS Console windows support a popup menu activated by the right mouse button. The popup menu depends upon the item selected in the window. In the case of the main window, the popup menu includes the following subset of commands from the menu bar:

Engine menu Reconfigure, Delete, Start, Logging, Trace, Event Trace, Status, Shut Down

Monitor menu Sessions, Processes Summary, Processes Resident, Activity Queues, Activities Resident, Registrations, Performance

Online Help

Online Help is available from iIS Console.

- To display the Help Topics window, choose Help > Help Topics.
- To display help for the current window, press the F1 or Help key (depending on your operating system platform).

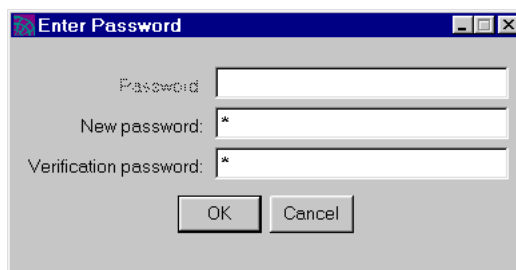
Setting Password Protection for iIS Console

You can restrict access to iIS Console by setting password protection for your active iIS environment. A password is then required to open iIS Console, as well as any other system management tools, such as Conductor Script and the iPlanet UDS Environment Console.

Once you set a password, other users cannot open iIS Console (or Conductor Script or the iPlanet UDS Environment Console) in your active environment without using the password.

► To set a password for iIS Console

1. Choose Environment > Set Password. The Enter Password window displays.



2. Enter a password and then enter it again in the Verification password field.
3. Click OK to set the password.

You can also use this window to change password protection for an environment by first entering the current password in the Password field and then entering the new password. If you want to remove password protection, enter the current password in the Password field, then enter nothing for the New password and click OK.

Exiting iIS Console

To exit the iIS Console, choose Environment > Exit.

Using iIS Console Windows

The iIS Console is a graphical system application developed in iPlanet UDS whose windows behave in accordance with your host window system. If you use graphical applications in your host window system, the Console will be familiar.

Using the Mouse

The iIS Console behaves like any standard application in your window system—mouse clicks select objects, double-clicks open objects, and click-and-drag operations move or copy objects. If your mouse has more than one button, you use the left-most button for these operations (or, if your mouse is configured specifically for left-handed use, the right-most button).

The iIS Console also uses the right-most button to present popup menu commands as an alternative to using the menu bar. The popup menu choices presented by right-clicking depend on which, if any, items are selected in the active window.

Window Refresh Behavior

Many iIS Console windows display lists that are maintained by the engine and are in a constant state of change. iIS Console lets you determine if and how often window displays are refreshed to correspond to information maintained by the engine. Each display window has a default automatic refresh time interval setting.

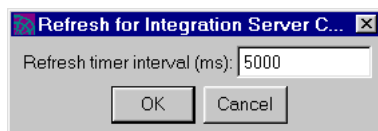
You can turn automatic refresh off or on, set the automatic refresh interval, or force an immediate refresh.

➤ **To turn automatic refresh off or on**

1. Enable the Automatic Refresh toggle in the bottom right of the window.

► **To set the automatic refresh interval**

1. Choose View > Set Refresh Interval. The Refresh Interval window displays with the default time interval setting:



2. Enter a new time interval in milliseconds.
A time interval of zero turns off automatic refresh.
3. Click OK.

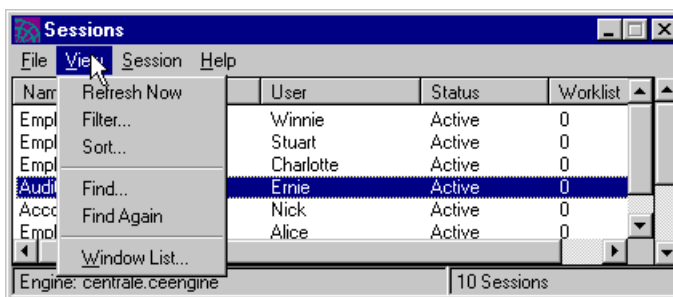
► **To force an immediate refresh**

1. Choose View > Refresh Now.

Filtering iIS Console Lists

Many iIS Console windows provide lists you can filter using commands available on the View menu and in the right mouse button popup menu. This section describes techniques used to perform these filtering operations.

For example, suppose you select an engine and then choose Monitor > Sessions to display a list of sessions for that engine. You can filter the list displayed or find sessions that meet certain criteria using the Filter and Find commands.



Selecting Filter from the View menu or the mouse menu displays a window in which you can write an expression for filtering the session list. The expression must be of the form:

```
Column_name <operator> value
```

Operators

You can build filter expressions that use both comparison and logical operators. The comparison operators are listed in the following table:

Comparison Operator	Description
=	equal to
<>	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

You can use a number of logical operators to join several filter expressions. The logical operators are listed in the following table:

Logical Operator	Description
and	logical and
or	logical or
not	logical not

Specifying Values

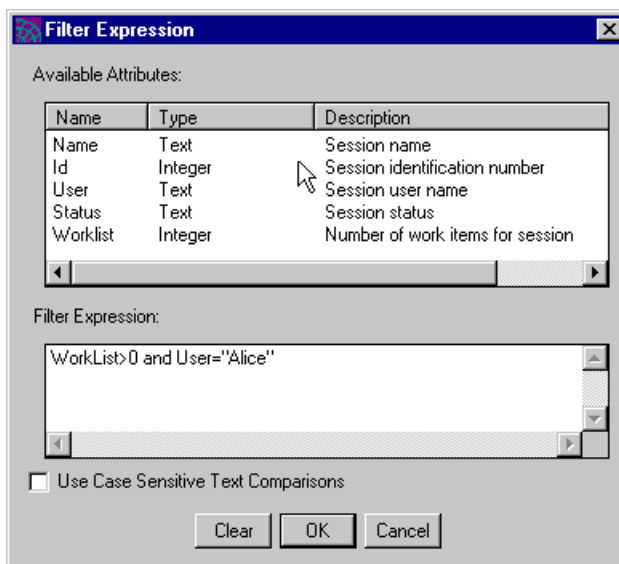
In a filter expression you specify the value of an object's state or status. These are expressed as case-sensitive string values and should be enclosed in double quotes:

Object	State/Status Value
Session	Active
Session	Suspended
Activity	PENDING

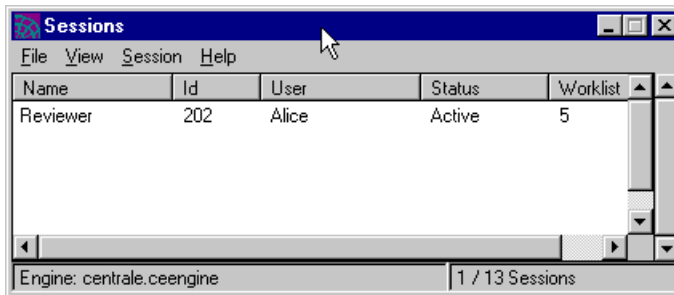
Object	State/Status Value
Activity	READY
Activity	ACTIVE
Activity	COMPLETED
Activity	ABORTED
Timer	ON
Timer	OFF

Example Filter Expression

In the session list example, suppose you enter the following expression in the Filter window:



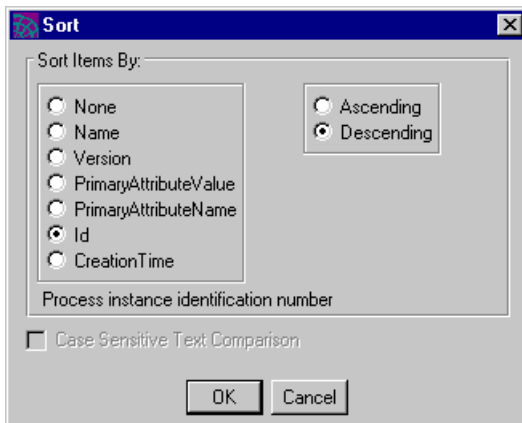
The resulting session list would then be the following:



NOTE Choose Clear to revert to the default list.

Sorting iIS Console Lists

The iIS Console lets you sort sessions, activities, and process instances by name, version, primary attribute value, primary attribute name, ID, or creation time.



1. From the Monitor menu select the type of information you want to sort. For example, if you want to sort process instances, choose Monitor > Processes Resident.
2. In the Processes Resident window that opens, choose View > Sort.
3. Select the item to sort by, and indicate ascending or descending order.

4. If you select a text field, you can also specify that case-sensitivity apply to the displayed text by checking Case Sensitive Text Comparison.
5. Click OK.

NOTE The items you can sort by varies according to the monitor window whose items you are sorting.

On Windows platforms, you can also specify an ascending sort by clicking a column name. For a descending sort, press Control while clicking the column name.

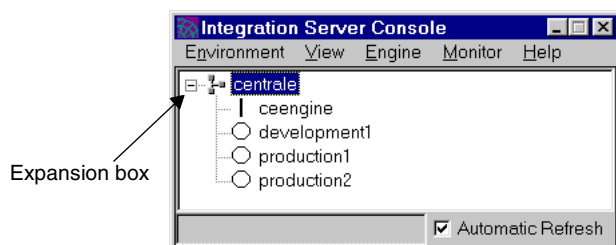
Using List Views

The iIS Console provides iPlanet UDS list views in the main window and the Engine Status window so you can browse engines and engine components.

You might be familiar with list views as the file directory display mechanism in your host window system. iPlanet UDS list views behave like the File Manager application in Microsoft Windows. This program displays hierarchical information successively indented from left to right. To select an element in a list view, you click it. To open an element to get a list of sub-elements, you double-click the element, then either press Enter while it is selected or click its expansion box.

The expansion box lets you open or close the list of elements “below” the current element in the hierarchy. If the expansion box displays a + (plus), click the box to open the list. If the expansion arrow displays a – (minus), click the box to close the list.

Figure 3-2 iIS Console Browser



iIS Console Main Window Command Summary

The following tables summarize the commands available from the iIS Console main window.

Environment Menu

Command	Description	See...
Alarms...	Lets you view all alarms received from the selected engine.	page 208
Registrations...	Provides submenus for registering (and unregistering) process definitions, assignment rules, user validations, and user profiles with one or more engines.	page 144
Aliases...	Provides submenus for registering (and unregistering) aliases with one or more engines. Aliases are used in specifying subprocess activities.	page 148
Set Password...	Opens a password entry window where you can set password protection for opening iIS Console in your active environment. Once you set a password, other users cannot open the iIS Console (or Conductor Script) in that environment without using the password.	page 71
Print...	Prints the iIS Console main window.	—
Print Setup...	Displays the Page Setup window for the window system.	—
Exit	Closes the iIS Console and all open windows.	page 71

View Menu

Command	Description	See...
Refresh Now	Forces an immediate refresh of the engine list in the main viewing panel.	page 72
Set Refresh Interval...	Opens a Refresh Interval window where you can set the time interval between automatic refreshes of the window display.	page 72
Window List...	Displays a list of open windows from which you can select a window to bring to the foreground.	—

Engine Menu

Command	Description	See...
New...	Opens a new engine configuration window in which you specify the component partitioning and static properties of a new engine.	page 85
Reconfigure...	Opens an engine configuration window for the selected engine in which you can modify the component partitioning and startup properties as specified in the engine configuration file.	page 113
Duplicate...	Creates a duplicate of the selected engine. A duplicated engine contains the same configuration as the original engine.	page 104
Delete	Lets you delete an engine configuration for the selected engine.	page 104
Start...	Opens an engine startup window in which you can start the selected engine or any of its components.	page 105
Logging...	Opens a dynamic database reconfiguration window in which you can dynamically change which database tables are being logged (current state or history log or both) as well as change the history log volume. The changes you make in this window do no change the startup configuration.	page 115

Command	Description	See...
Trace...	Opens an engine trace window in which you display messages, of the type you specify, generated by the various engine components.	page 212
Event Trace...	Opens the engine event filter window which lets you select the type of filter to use for displaying engine events as well as a filter time interval.	page 217
Status...	Opens an engine status window in which you can view the status of the selected engine or any of its components. You can also change the state of engine units.	page 117
Shut Down	Shuts down the selected engine and all its components.	page 124

Monitor Menu

Command	Description	See...
Sessions...	Opens the engine sessions window, which displays the list of active sessions for the selected engine. You can monitor and manage sessions from this window.	page 173
Processes Summary...	Opens the processes summary window, which displays the list of all process definitions for the selected engine and the number of instances of each. You can open a named process definition to get a list of the process instances, displaying information about each.	page 221
Processes Resident...	Opens the engine process instances window, which displays the list of all process instances for the selected engine. You can sort this list by process name, primary process attribute, process_id, or creation time. You can open a listed process instance to get information about its activities, attributes, timers, and so on.	page 179
Activity Queues...	Opens the engine activity queues window, which displays the list of all activity queues for the selected engine. Activity queues are always resident in the engine. You can open a listed queue to get a list of activities in the queue, and so on.	page 179

Command	Description	See...
Activities Resident...	Opens the engine activity instances window, which displays the list of all activity instances for the selected engine. You can sort this list by activity name, activity ID, activity state, and process name. You can open a listed activity instance to get information about its type, description, sessions whose activity list it is on, process attributes, timer links, and so on.	page 179
Registrations...	Opens the engine registrations window, which displays the list of process definitions, assignment rules, user profiles, user validation, and aliases currently registered with the selected engine. This window can be the launch point for monitoring and managing execution of processes registered with the engine—for example, you can monitor the status of all process instances of a given named process. You can also deregister a registered item from the selected engine.	page 150
Performance...	Opens the performance window, which displays histograms of four agent data types (instruments) that characterize the performance of the selected engine.	page 221

Help Menu

Command	Description	See...
Contents	Opens the Help Topics window that lets you navigate the iIS process management online help system.	page 70
About iIS...	Displays information about the iIS product.	—

Managing Engines

This chapter describes how to manage iIS process engines. It covers the following topics:

- configuring an engine
- starting an engine
- monitoring and changing the state of an engine
- reconfiguring an engine to accommodate increased load, improve performance, or recover from failure
- managing an engine database

Production Engines Versus Development Engines

Engine use in production environments can differ significantly for use in development environments. In a development environment, the engine is used principally for testing and debugging purposes, while in a production environment, the engine must support mission-critical operations.

These different use requirements translate into differences in the way you are likely to configure and manage an engine. The table below summarizes some of the issues.

Production Engine	Development Engine
Multiple engines might be needed for organizational reasons or to handle heavy production loads.	A single engine is normally sufficient for testing purposes.

Production Engine	Development Engine
State information is needed for recovery: current state logging to the engine database is always turned on.	State information does not need to be stored in the engine database: current state logging can be turned off.
Backup engine unit and governor is needed for failover.	Failover is not critical: a minimal engine configuration is adequate.
A number of database services are normally required to meet engine performance (database access) requirements.	A single database service is normally adequate.
Historical process execution information is needed to analyze and improve system performance: logging of historical state information is normally turned on.	Current process execution information is more important than historical information: history logging can be turned off.
Engine is rarely cold started: registration and history information would be lost.	Engine is often cold started: registration information is purged, allowing for a clean engine state.
Application upgrades need to be performed without interrupting process execution.	Engine can be shut down and cold started to perform application upgrades.
Resource limitations need to be monitored to ensure engine performance and stability: memory, disk space, and cpu utilization.	Resource limitations rarely are of concern in development environments.

Keep these issues in mind when configuring an engine. In most situations, you have to configure and manage at least one development engine and one production engine.

NOTE Do not use the same engine for both development and production.

Configuring an Engine

An iIS process system can have any number of engines. For example, one engine might be used for testing and another used for production. In other cases, organizational considerations or heavy load conditions might require a number of production engines. While engines can communicate with one another—a process executing on one engine can be invoked from a process executing on another—each engine operates independently. One engine, for example, cannot fail over to another engine.

Each engine is composed of a number of engine components—engine units, governor, and database services (see [Chapter 1, “Introduction: iIS Process Management”](#)). For each engine to function independently, its components must each be identified with the engine and have a unique name, provided at startup time.

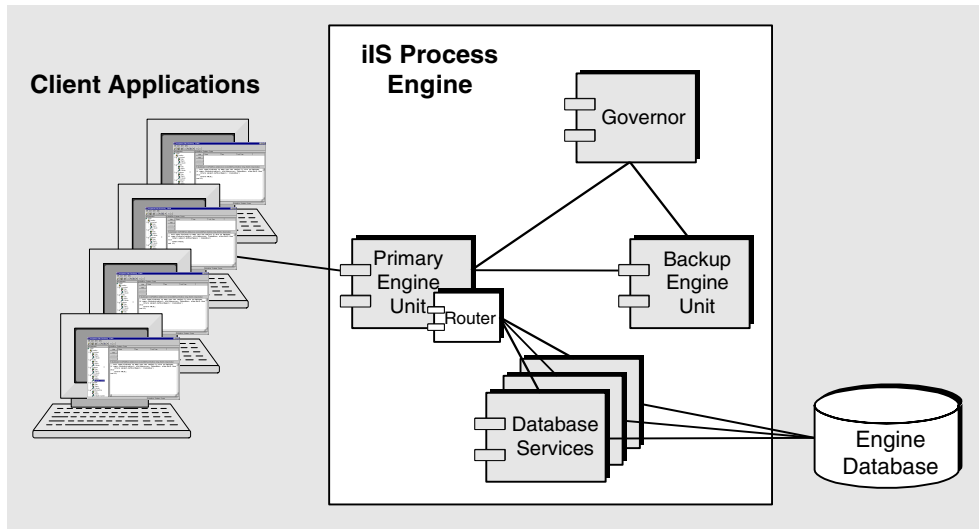
Each engine requires two levels of configuration:

- engine component partitioning
Specify which computer nodes in your environment will host the different engine components—engine units, governor, and database services.
- engine startup properties
Specify a number of startup properties required by the various engine components.

The two levels of configuration are considered separately, below.

Engine Component Partitioning

An engine that is fully configured for failover and load balancing includes all the engine components shown in [Figure 4-1](#), described on [page 35](#).

Figure 4-1 Engine Configuration with both Failover and Load Balancing

During installation, you decided which nodes in your iPlanet UDS environment (engine servers) would have engine components installed on them. You must also determine your engine component partitioning: which engine components (if any) will run on each engine server.

The partitioning scheme that provides the highest failover reliability and the highest performance depends on a number of different factors. These factors include the power and reliability of your servers, the bandwidth of your network, the normal load on the iIS process engine, and how this load is distributed over a typical work day. Since these factors vary from site to site, only some very general guidelines are discussed here.

Primary Engine Unit It is recommended that you assign the primary engine unit to a fast, reliable server node. The primary engine unit in a production situation normally maintains sessions with hundreds of clients, maintains state information on tens of thousands to millions of objects, and performs all the computations required of an engine.

Backup Engine Unit It is recommended that you assign the backup engine unit to a capable server node also, but a different one from the node on which the primary engine unit runs (to provide backup should the primary server fail). This node could be a less powerful computer if you are willing to accept a possible decrease in performance for your backup unit.

Governor The governor does not do much processing and does not require a high performance server. It is recommended that you *not* assign it to the same server as either the primary or backup engine unit, because it would then not be able to distinguish between failure of the primary engine unit and a break in the network link between the engine units.

Database Services The number and placement of database services depends on load conditions. You must have enough database services running to prevent database access requests from backing up in the database service router (see [“Viewing Performance Indicators” on page 222](#)). In general, assign high priority database services to a very capable server node, probably the same as the one on which the database manager resides. If your database system provides network access to the database, such as ORACLE’s SQL*Net, then lower priority database services can be assigned to under-utilized servers, where resources are available to help carry heavy loads on the engine. (If your database system does not provide network access to the database, then your database services must reside on the same node as your database manager.)

A typical configuration is illustrated in [Figure 4-2 on page 88](#).

In this illustration the database services on Server4 are used in a round-robin fashion unless the load on the engine is too heavy for them to carry it. In that case the database service on Server1 is used. The database service on Server3 will be used to handle peak loads, but will get less work than the service on Server1.

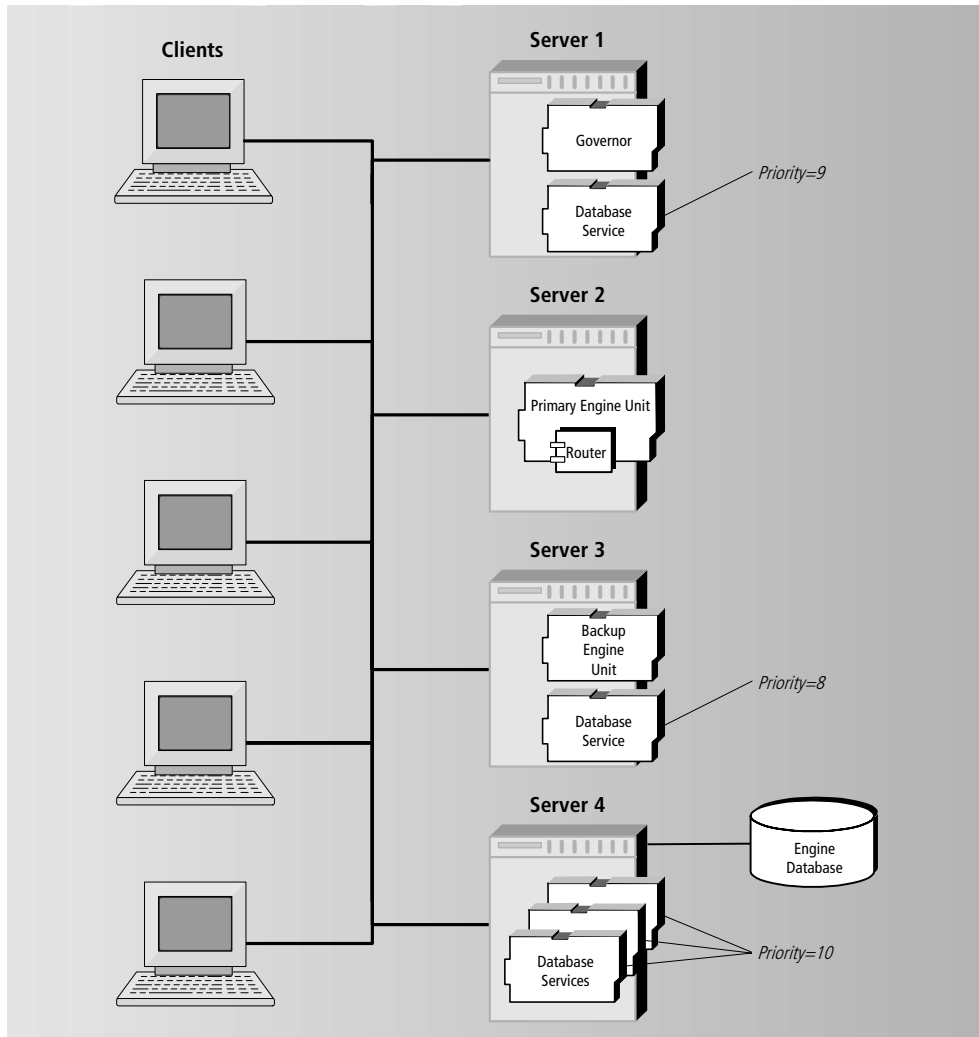
You did not have to decide at iIS installation time exactly which engine components would be assigned to each server node because all component partitions are installed on each server node that supports an engine. Instead, you must select which components to start on which nodes when you configure each engine.

For example, you can designate a default set of database services that are started at engine startup time. Subsequently, however, you can start up additional database services on designated nodes, as needed, to accommodate increasing load on the engine.

The location of the primary and backup engine units is best determined in advance, before startup time, and should not change because the shared library files corresponding to process definitions registered with the engine are needed on all the nodes on which the engine units are located. For this reason, moving the engine units from one node to another can take a long time.

Your default engine component partitioning scheme is stored in an engine configuration file, allowing you to design your basic engine component partitioning scheme in advance and consistently start up your engine the same way every time.

Figure 4-2 Typical Engine Partitioning Scheme



Engine Startup Properties

Before you start an engine, you must specify a number of startup properties. These properties are stored in the engine configuration file. The startup properties are the following:

Engine name An alpha-numeric name used to identify the engine in a given environment.

Environment An alpha-numeric name that specifies the iPlanet UDS environment in which the engine resides.

Database Configuration Properties Properties needed to access the engine database. These properties include:

- database type
- database name
- user name
- user password

Database Logging Settings These settings specify whether state information is logged and which history information tables are updated by the engine.

Process Engine Components A minimal engine configuration contains an engine unit component and a database service component. An engine configured for failover also contains a Governor component.

Memory Settings These settings specify the object memory heap size allocated to the various engine components (governor, database services, and engine units). Included here is also a setting to limit the number of console monitor windows that can be open at any time.

Stack Size These settings specify the memory size of each thread started for the process engine components. Default values vary for each platform. Typically, you do not need to override the default values. For information on setting stack size in an iPlanet UDS environment, refer to the discussion on FORTE_STACK_SIZE in the *iPlanet UDS System Management Guide*.

Process Execution Options These settings specify how an engine manages process execution. By setting these options appropriately, you can ensure that memory resources do not become overtaxed during process execution, and thereby avoid engine performance degradation or possible failure.

Engine Recovery Options These options specify the behavior of the engine during recovery from an engine failure. They determine how long an engine waits for client sessions to reconnect, and the action to take if the client sessions do not reconnect after the timeout period expires.

Engine Configuration File

The engine component partitioning and engine startup properties are stored in an engine configuration file in the following location on the central server node in your environment:

FORTE_ROOT/sysdata/conductr/clN/environment/engine_name.cfg

Path Element	Description
environment	Name of the iPlanet UDS environment in which engine resides.
engine_name	Name of the iIS engine.

A sample configuration file is reproduced below:

```

FILE_VERSION = 2.0
GOVERNOR = Server4
UNITS = Server1:Primer0:P,Server3:Secundo:B
DBSERVICES = Server2:DBService1:10,Server4:DBService2:9,...
DATABASE_TYPE = ORACLE
DATABASE_NAME = @oracle_Server2
DATABASE_USER_NAME = wftester
DATABASE_USER_PASSWORD = *#@$!!
LOGGING= ON,STATE,LOG,LOG_ACTIVITIES,LOG_ATTRIBUTES,LOG_LOCKS,...
GOVERNOR_MEMORY_FLAG = n:2000,x:10000,...
DBSERVICE_MEMORY_FLAG = n:2000,x:10000,...
UNIT_MEMORY_FLAG = n:10000,m:30000,...
GOVERNOR_STACK_SIZE = 48000
DBSERVICE_STACK_SIZE = 96000
UNIT_STACK_SIZE = 96000
UNIT_RECOVER_CURSOR_SIZE = 123
DATABASE_PROC_SEGMENT_SIZE = 256
DATABASE_SESS_SEGMENT_SIZE = 256
UNIT_MAX-SESSION_NUMBER = 300
UNIT_MAX-PROCESS_NUMBER = 0
UNIT_IN_MEMORY_PROCESS_LIMIT = 1000
UNIT_SWAPOUT_INTERVAL = 10000
MONITOR_DISPLAY_LIMIT = 10
UNIT_SESSION_RECONNECT_LIMIT = 600000
UNIT_SESSION_RECONNECT_ACTION = 2048
UNIT_SNS_RATIO = 70

```

The engine properties specified in the configuration file are described in the following table:

Table 4-1 Process Engine Configuration Properties

Property	Type of value	Description
FILE_VERSION	string	Internal use. Do not modify.
GOVERNOR	string	Specifies the node on which governor component runs.
UNITS	string	For each engine unit, specifies the node on which the unit runs, unit name, and preferred priority (P=primary unit, B=backup unit, or N=no preference). String contains entries, separated by commas, in the form [NODE_NAME:UNIT_NAME:P/B/N][,...].
DBSERVICES	string	For each database service, specifies the node on which the service runs, service name, and priority (integer—positive or negative). String contains entries, separated by commas, in the form [NODE_NAME:SERVICE_NAME:PRIORITY][,...].
DATABASE_TYPE	string	Database type. Can have the following values: ORACLE, SYBASE, ... (any database system supported by iPlanet UDS).
DATABASE_NAME	string	Name of database.
DATABASE_USER_NAME	string	Database logon user name.
DATABASE_USER_PASSWORD	string	Database logon password. The password is encrypted.
LOGGING	string	Specifies which tables (current state and history log) are enabled and which object state changes are recorded if the history log is enabled. String contains any of the following entries, separated by commas, that are enabled: [STATE] [,LOG] [,LOG_ACTIVITIES] [,LOG_ATTRIBUTES] [,LOG_LOCKS] [,LOG_PROCESSES] [,LOG_SESSIONS] [,LOG_TIMERS]

Table 4-1 Process Engine Configuration Properties (Continued)

Property	Type of value	Description
GOVERNOR_MEMORY_FLAG	string	Specifies memory options. Uses the same syntax as the -fm flag used on iPlanet UDS partitions. Default values are n:2048,x:16384,....
DBSERVICE_MEMORY_FLAG	string	Specifies memory options. Uses the same syntax as the -fm flag used on iPlanet UDS partitions. Default values are n:2048,x:16384,....
UNIT_MEMORY_FLAG	string	Specifies memory options. Uses the same syntax as the -fm flag used on iPlanet UDS partitions. Default values are n:2048,x:16384,....
GOVERNOR_STACK_SIZE DBSERVICE_STACK_SIZE UNIT_STACK_SIZE	integer	Specifies the memory size for each thread started for the process engine components. Default values vary for each platform. For information on setting stack size in an iPlanet UDS environment, refer to the discussion on FORTE_STACK_SIZE in the <i>iPlanet UDS System Management Guide</i> .
UNIT_RECOVER_CURSOR_SIZE	string	Specifies number of process instances to be recovered at one time in the event of engine unit failure.
DATABASE_PROC_SEGMENT_SIZE	integer	Specifies segment size for storing text-based process attributes. Default value is 255 bytes.
DATABASE_SESS_SEGMENT_SIZE	integer	Specifies segment size for storing text-based session attributes (associated with a session's user profile). Default value is 255 bytes.
UNIT_MAX_SESSION_NUMBER	integer	Specifies number of concurrent sessions supported by the engine. Default value is 300.
UNIT_MAX_PROCESS_NUMBER	integer	Specifies the number of concurrent process instances supported by the engine. Default value is 0 (no maximum).
UNIT_IN_MEMORY_PROCESS_LIMIT	integer	Specifies the number of process instances retained in memory. Default value is 1000.
UNIT_SWAPOUT_INTERVAL	integer	Specifies the time interval (in milliseconds) at which the engine checks the number of process instances resident in memory and swaps out dormant process instances. Default value is 10000.

Table 4-1 Process Engine Configuration Properties (*Continued*)

Property	Type of value	Description
MONITOR_DISPLAY_LIMIT	integer	Specifies the maximum number of Cconsole monitor displays that can be simultaneously open by all Cconsole sessions monitoring the engine. This limit affects the Sessions, Processes Resident and Activities Resident monitor windows. The number of these windows that are open affects the performance of the engine.
UNIT_SESSION_RECONNECT_LIMIT	integer	During engine recovery, specifies the time (in milliseconds) that an engine waits for clients to reconnect to a session. Default value is 60000 milliseconds (10 minutes).
UNIT_SESSION_RECONNECT_ACTION	integer	During engine recovery, specifies the action to take (suspend or terminate) for client sessions that fail to reconnect before the specified timeout period (UNIT_SESSION_RECONNECT_LIMIT). Default value specifies to suspend sessions that do not reconnect.
UNIT_SNS_RATIO	integer	Specifies the ratio of swappable to non-swappable processes for the process engine. Processes that are not marked as recoverable (either through the process definition or by a process client calling CreateProcess) cannot be swapped out of memory by the engine. Adjust the default value only if you are having performance problems with the process engine.

Customizing Engine Database Schema

iIS provides a default database schema that it uses to create an engine database. This default schema can be used in a development environment without modification. However, for deployed applications, you can modify the database schema according to your own database requirements. Typically, you modify the schema to add table space qualifiers or other database-specific qualifiers to the CreateTable and CreateIndex statements.

When you configure a new engine, the default engine schema is specified by the following file on the central server node in your environment:

FORTE_ROOT/sysdata/conductr/clN/environment/engine_name.dbs

Path Element	Description
environment	Name of the iPlanet UDS environment in which engine resides.
engine_name	Name of the iIS engine.

The database schema file uses XML to define SQL statements for the schema. When you perform a cold start on an engine, iIS uses the specification in this file to create the engine database.

If you want to modify the default schema, before performing a cold start on an engine, edit this file according to your needs.

CAUTION When modifying an engine database schema, do not change any basic table definitions or index definitions, and do not change any column names or column data types. You should only modify information specific to a database or site, such as the table space qualifiers. Otherwise, the engine may not start. Examine the engine log file (at *FORTE_ROOT/log*) to determine the cause of startup failure.

The following example shows how to modify the create index statement for the WFHActivity table to include a table space qualifier.

Code Example 4-1 Generated Definition (Oracle)

```
<FNCreateIndexStatement>
  create unique index wfk_act on WFHActivity (
    processid,
    id)
</FNCreateIndexStatement>
```

Code Example 4-2 Modified Definition

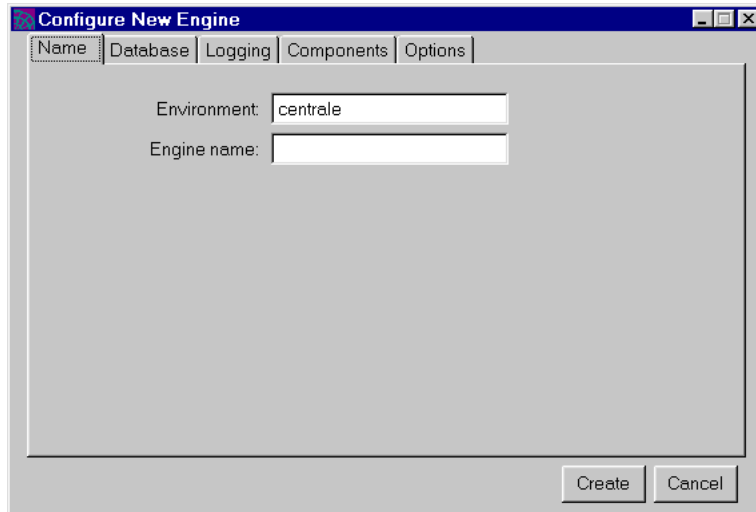
```
<FNCreateIndexStatement>
  create unique index wfk_act on WFHActivity (
    processid,
    id)
    TABLESPACE my_ProcessEngine_Index
</FNCreateIndexStatement>
```

How to Configure a New Engine

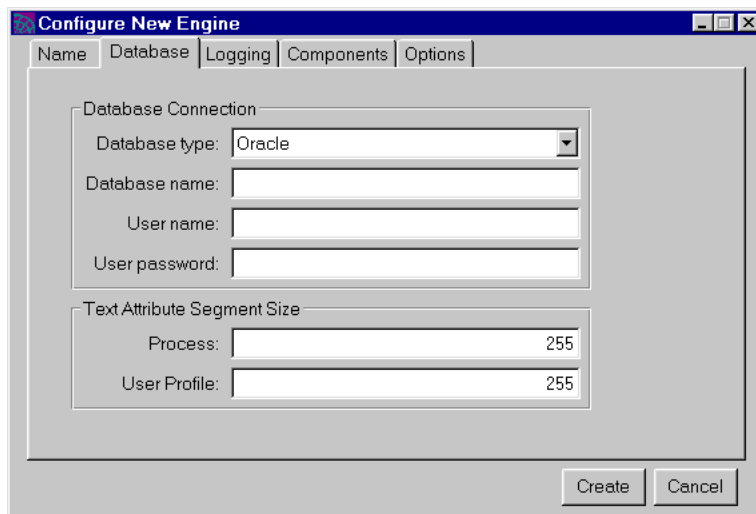
The easiest way to configure a new engine is to use the iIS Console to create a new engine configuration file and place it in the proper directory location on your central server node. (You can also create a new configuration file by directly entering the information in [“Engine Configuration File” on page 90](#) into the file.)

► **To configure a new engine**

1. Choose Engine > New. The Configure New Engine window displays:



2. Enter a name for the new engine in the Engine name field.
The name can be case-sensitive and of any length, but have no spaces.
3. Click the Database tab to display the database properties dialog:

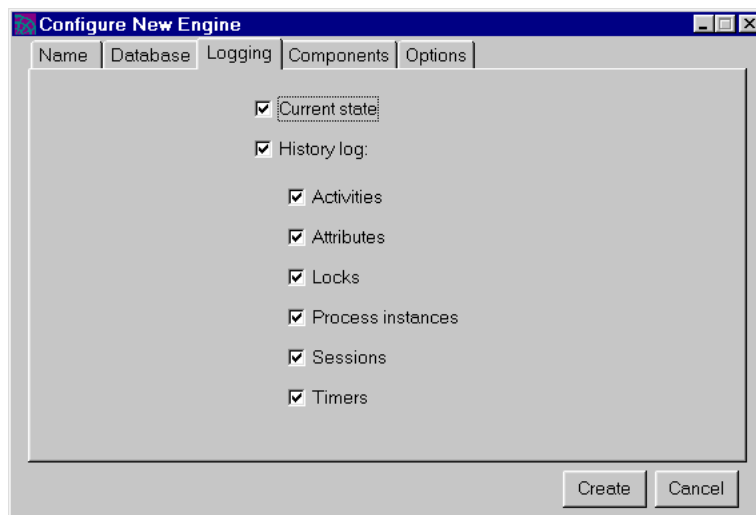


4. Enter the database configuration properties.

For information on the Database Connection fields, refer to the iIS online help. The Text Attribute Segment Size fields allows you to specify the segment length for process attributes and user profile information for a session.

Property	Description
Process attribute segment size	Specifies segment size for storing text-based process attributes. Default value is 255 bytes.
User profile attribute segment size	Specifies segment size for storing text-based session attributes (associated with a session's user profile). Default value is 255 bytes.

5. Click the Logging tab to display the logging properties dialog:



6. Specify the engine database tables (Current state and/or History log) for which you want to log information. If you write to history log tables, specify the information to log.

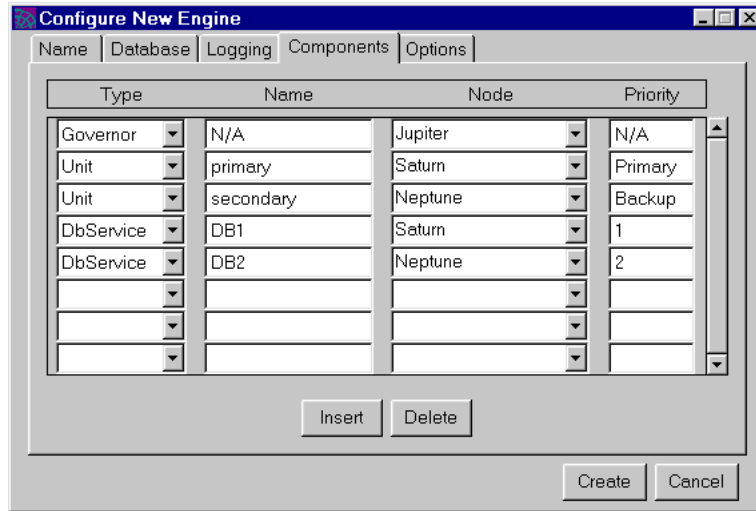
Current state This setting determines whether the engine writes information about the current state of all sessions, processes, and process components to the current state tables (see [Appendix B, “Engine Database Schema”](#)). You cannot recover an engine or support failover of an engine unit unless you enable the Current state option.

History log This setting determines which information, if any, the engine logs about each state change in sessions, processes, and process components to the history log tables (see [Appendix B, “Engine Database Schema”](#)). The history log tables are used for obtaining historical information about process execution. The history log can grow quite large, so set logging selectively. You may have to monitor, back up, and flush the history log tables on a regular basis. For more information, see [“Managing an Engine Database” on page 124](#).

NOTE The registration tables are always enabled for registering process definitions, assignment rule dictionaries, user profiles, a validation, and aliases. Therefore, no option for the registration table appears in the dialog.

7. Select the Components tab.

The Component Partitioning dialog is displayed.



8. Specify the engine component partitioning.

When assigning a component to a node, be sure that the node is an engine server node (that is, that engine components were installed on the node during iIS installation). The Node drop list shows all server nodes (whether online or not) defined in the iPlanet UDS environment.

The order in which you specify components is not important.

Governor To provide failover for an engine, you need to include a governor in the configuration. The governor does not accept a name or priority (indicated by “N/A” in the table).

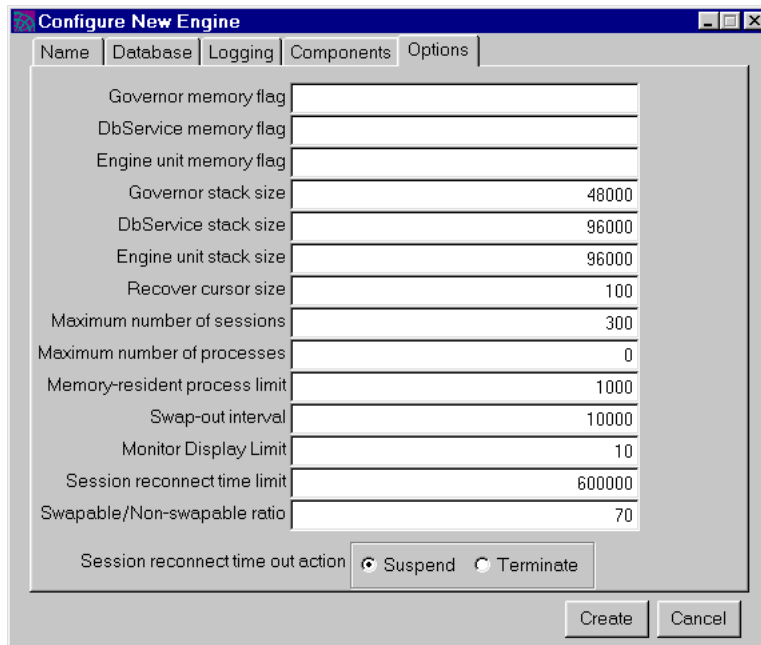
Engine Unit You can specify one or two engine units in your configuration—to provide failover, you need two units (a primary and a backup). For each engine unit, you must provide a name and indicate whether it is the primary unit (P), backup unit (B), or not specified (N). In the last case the governor decides which is primary and which is backup. For failover to work, the engine units must be put on separate nodes and cannot be on the node hosting the governor.

Database Service You can specify as many database services as you need (at least one) to balance the maximum load on your engine database access. For each database service, you must provide a unique name and indicate its priority. A priority is simply an integer (positive or negative from 1 to 10), with

a higher numeric value signifying a higher priority. It is suggested that you decide on a sensible set of priorities. A default priority of “1” is used if you do not specify a priority. The engine distributes the database access load based on this priority. See [“Full Configuration: Failover and Load Balancing Combined” on page 35](#).

Database services must be placed on the node where your database manager resides, unless your database system provides network access to the database, such as ORACLE’s SQL*Net. If your database system provides network access to the database, then lower priority database services can be assigned to under-utilized servers where resources are available to help carry heavy loads on the engine. Any node that hosts a database service must be specified in the iPlanet UDS environment definition as supporting a database resource manager (otherwise, a database service will not be installed on the node).

9. Click the Options tab to display the Options dialog:



10. Specify memory options for the governor, database service, and engine unit memory flags if you want to override the default values.

The settings you are most likely to change are the minimum and maximum memory allocation for engine units. The minimum allocation specifies the default size of the object memory heap and the maximum allocation sets the upper limit to which the iPlanet UDS memory manager can expand the object memory heap. The default values might prove insufficient for engines that are processing thousands of process instances, each with many activities, attributes (and locks), timers, sessions, and so forth.

To specify memory options, use the following syntax:

```
memory_option { : | = } number [, memory_option { : | = } number]
```

Do not include any spaces.

For example, n:4096, x:16384 specifies both maximum and minimum sizes of the iPlanet UDS memory heap. The n and x memory options are described in the following table (a page is 1024 bytes of memory).

Memory Option	Description
n	Minimum number of pages managed by the memory manager. The value specifies the absolute minimum number of pages that will be allocated to the memory heap. Range is 1024 to 4194304 (32384 on Windows 3.1). Must be less than the x memory option. The default value is 2048.
x	Maximum number of pages managed by the memory manager. The value specifies the absolute maximum number of pages that can be allocated to the memory heap. Range is 1024 to 4194304 (32384 on Windows 3.1). Must be greater than the n memory option. The default value is 16384.

For more information about the iPlanet UDS memory manager and other memory options, see the *iPlanet UDS System Management Guide*.

11. Specify how many monitor windows can be open at any time.

The Monitor Display Limit option (default value of 10), indicates how many iIS Console monitor windows can be open at any time. This limit applies to Sessions, Processes Resident and Activities Resident monitor windows and affects all iIS Consoles monitoring the engine. The number of monitor windows open affects the performance of an engine.

NOTE If you attempt to open additional windows beyond the limit, an error message is displayed.

12. Specify the recover cursor size.

You can set the number of process instances to be recovered at one time in the event of engine unit failure. The default is 100 process instances at a time. The cursor size controls how many rows are accessed from the engine database. Lowering the number of instances (by lowering the cursor size) reduces the amount of memory needed for recovery; the trade off is a slower recovery. If your resources permit, you can increase the number for maximum efficiency.

NOTE If the engine unit fails to recover due to an out-of-memory error, you can lower the cursor size. Another option is to increase the maximum memory allocation for DB services or engine units as your resources permit.

13. Specify process execution options.

You can set a number of configuration options that affect how an engine manages process execution. By setting these options appropriately, you can ensure that memory resources do not become overtaxed during process execution, and thereby avoid engine performance degradation or possible failure.

The process execution options are the following:

Process Execution Option	Description
Maximum number of sessions	The maximum number of concurrent sessions supported by the engine. Attempts to open additional sessions will raise an exception. The default value is 300.
Maximum number of processes	The maximum number of concurrent process instance executions supported by the engine. Attempts to instantiate additional process instances will raise an exception. The default value is 0 (no maximum)
Memory-resident process limit	The number of process instances retained in memory. When the number of memory-resident process instances exceeds this limit, the engine swaps out dormant process instances, allowing for execution of newly activated process instances. The default value is 1000.
Swap-out interval	The time interval (in milliseconds) at which the engine checks the number of process instances resident in memory and swaps out dormant process instances. The default value is 10,000 milliseconds.

For information about how to use these configuration options, see [“How to Tune Process Execution” on page 116](#).

14. Specify engine recovery behavior.

If an engine goes down, and then recovers, you can specify how long the engine waits for clients to reconnect and the behavior of the engine in the event the client does not reconnect before the timeout period.

By default, an engine waits 10 minutes (600000 milliseconds) for a client to reconnect. You can modify this value from the Options tab. If a client fails to reconnect to the engine after the timeout period, by default, the engine suspends the session. You can change the default behavior, and specify that the engine terminates the session.

15. When you have finished configuring your engine, click the Create button to create the configuration file.

The engine configuration file is saved to a standard location on the central server (see [“Engine Configuration File” on page 90](#)).

Duplicating an Engine Configuration

If you have multiple engines that need to be configured similarly, or want to add an engine with a configuration similar to an existing engine, you can duplicate an engine configuration. After creating a duplicate engine you can reconfigure the new engine to make any necessary adjustments.

You can do this from the iIS Console main window.

➤ **To duplicate an engine configuration**

1. From the iIS Console main window, select the engine you want to duplicate.
2. Choose Engine > Duplicate.
3. Enter a new name for the duplicate engine.

The name can be case-sensitive and of any length, but have no spaces.

The duplicate engine shows up in the engine list in the iIS Console main window. You can reconfigure the engine to make any necessary adjustments (see [“How to Reconfigure an Engine” on page 113](#)).

Deleting an Engine Configuration

You can delete an engine from the list of configured engines in your environment by deleting the corresponding engine configuration file. You can do this from the iIS Console main window.

➤ **To delete an engine configuration**

1. Select an engine from the engine list in the iIS Console main window.
2. Choose Engine > Delete.

You can also delete the configuration file manually. Its location is shown in [“Engine Configuration File” on page 90](#).

Starting an Engine

To start an engine, you start each of its components—engine units, governor, and database services. These components are applications, each consisting of a single server partition that must establish communication channels with other components.

As each of these partitions starts, it registers itself with the iPlanet UDS Name Service and then looks in the Name Service registry for the names of other components with which it must establish a communication channel. The components can start in any order: they simply wait for the other components that they connect to. When all necessary communication channels are established, the primary engine unit can log on to its database and perform the operations that bring it online.

Each of the engine components and its start characteristics are discussed in more detail below. This background information is followed by instructions on how to start the engine using the iIS Console. For information on starting engines using Conductor Script, see [“Starting an Engine” on page 235](#).

Governor

When the governor starts, it does not try to connect to anything, but enters a state in which it is not connected to any engine units and waits for engine units to make contact. When the engine units start up, the governor determines which unit is to be primary and which is to be backup. There are three cases:

- If the first engine unit is marked as preferred primary, then the governor sets it as the primary unit.
- If the first engine unit is marked as preferred backup, then the governor sets it as the backup unit.
- If no engine unit is marked as preferred primary, then the governor sets the first to start as primary unit.

If the governor starts up *after* the engine units and they have independently established which is primary and which is backup, the governor accepts these settings.

The governor has four internal states, depending on its connections with the engine units:

Governor state	Description
IDLE	Not connected to any engine units
E1	Connected only to the first engine unit to start up (EngineUnit1)
BOTH	Connected to both engine units
E2	Connected only to the second engine unit to start up (EngineUnit2)

As the governor changes state (that is, as engine units start up and shut down), it determines which engine unit is primary based on the state of the engine units before the change occurred. For example, in going from BOTH to E2, the governor will make EngineUnit2 primary if that unit had been the backup before the state change (a normal failover scenario). In going back, from E2 to BOTH, however, it will retain EngineUnit2 as primary, even if that engine unit is marked as the preferred backup. (You can override the Governor to return EngineUnit1 to primary—see [“Changing Engine States” on page 122.](#))

Engine Unit

When an engine unit starts, it attempts to connect to both the governor and its partner engine. It waits for a timeout period and exits if no connection is forthcoming. (If the engine configuration file specifies no engine unit partner, the engine unit can start up standalone, without requiring a governor or partner.)

If an engine unit starts *after* the governor has started, the engine unit connects to the governor. The governor determines if the engine unit is primary or backup, depending on the engine’s preferred designation or whether the engine is the first unit to start (in which case it becomes the primary unit). If the engine unit is primary, it attempts to come online. If the engine unit is backup, it enters a standby state in which it does not attempt to come online.

If an engine unit starts *before* the governor, but after its partner has started, then it must negotiate with its partner to establish which unit is to become primary. If one unit has been marked as the preferred primary, then that unit becomes primary. If neither unit has been marked as the preferred primary, then the negotiation randomly determines which of the two becomes primary. If the governor starts up in the middle of this negotiation, the governor takes over and decides the issue. If the governor starts up after the negotiation is complete, the governor accepts the negotiated decision.

When an engine unit starts, it goes through an internal, transitional state (EMBRYONIC), then assumes one of two persistent states, ONLINE or STANDBY, depending on whether it is designated as the primary or backup engine unit. The STANDBY state is assumed immediately upon startup; however, the ONLINE state is achieved only after the engine unit has passed through a number of startup phases, such as logging on to the engine database and performing registrations with the Name Service. The startup phases are described in more detail below.

As a system manager you can override the state of an engine unit after it has started. For example, you can set the primary engine to a STANDBY state, and then set its partner to an ONLINE state, making the partner the new primary engine unit. The engine unit states are described below.

Engine unit state	Description
EMBRYONIC	Internal, transitional startup state.
ONLINE	Normal state of the primary engine unit. It is running, connected to the engine database, and can accept client sessions.
STANDBY	Normal state of the backup engine unit. It is running, but is not connected to the engine database and cannot accept client sessions.

Primary Engine Unit Startup Phases

The primary engine unit must perform a number of startup operations to come fully online. These operations depend on whether the primary engine unit is recovering from failure, is restarting an engine on an existing database, or is starting up for the first time (cold start). The primary engine unit startup phases are documented in the following table to assist you in diagnosing problems in engine startup (the phases are listed in sequential order):

Startup Phase	Description	Phase #
REMOVE_PRIMARY	Removes any existing “primary” registration in the Name Service. This phase not included in a cold start.	51
CREATE_ENGINE	Instantiates an engine object.	52
ATTACH_HISTORY_MANAGER	Beginning of the database logon phase.	53
INIT_HISTORY_DB	Database logon phase, in which a router is created and registered with the Name Service, a communication channel with a database service is opened, and a database service logon to the database is successfully completed.	54
HISTORY_INITIALIZED	Completion of the database logon phase.	55
COLD_DELETE	For a cold start, deletes any existing database tables.	56
COLD_CREATE	For a cold start, creates the required database schema (tables).	57
RECOVER_OBJECTS	Recovers all current state objects from the database and places them in engine state memory. This phase not included in a cold start.	61
REGISTER_PRIMARY	Places a “primary” registration in the Name Service. This allows all other objects to locate the primary engine unit.	62
INFORM_GOVERNOR	Communicates to governor that state has changed to online.	63

Startup Phase	Description	Phase #
INFORM_PARTNER	Communicates to partner that state has changed to online.	64
STARTUP_COMPLETE	Completion of startup operations. Engine unit now online.	65

Database Service

When a database service starts, it attempts to connect to the primary engine unit. If the primary engine unit is not online, the database service continues waiting for it indefinitely.

When the primary engine unit starts up, the database service opens a communication channel to the engine's database service router object and becomes available for use. The database service first asks the router for the name of the database along with the username and password to use for logging on (provided to the engine unit in the engine unit configuration file). It then logs on to the given database using the username and password provided.

When you start a database service, you specify a component name and a priority for it. A priority is simply an integer (positive or negative), with a higher numeric value signifying a higher priority. It is suggested that you decide on a sensible set of priorities. A default priority of "1" is used if you do not specify a priority.

When a database service starts, it may (or may not) establish a communication channel with a router, and it may (or may not) log on to the engine database successfully. The database service, therefore, can assume the states shown in the following table. If a database service detects a failure in the communication channel to the primary engine unit, it logs off the database and reconnects to a new primary unit when its router comes online.

Database service state	Description
RouterChannel=OPEN/CLOSED	Communication channel with the database service router is open or closed.
DatabaseState=TRUE/FALSE	Database logon is successful or unsuccessful.

New database services can be started at any time, and they automatically become available to the primary engine unit when they connect.

How to Start an Engine

You can start engine components from a centralized location using the iIS Console (see [Chapter 3, “The iIS Console”](#)). The iIS Console uses iPlanet UDS system management agents to start the partitions corresponding to each of the engine components, as specified in the engine configuration file.

► **To start an engine**

1. Select an engine in the iIS Console main window.
2. Choose Engine > Start. The Start Engine window displays:

The 'Start Engine' dialog box contains the following fields and table:

Environment:

Engine:

Startup option:

Components

State	Type	Name	Node	Priority	
<input checked="" type="checkbox"/>	<input type="radio"/>	Governor	N/A	Jupiter	N/A
<input checked="" type="checkbox"/>	<input type="radio"/>	Unit	primary	Saturn	Primary
<input checked="" type="checkbox"/>	<input type="radio"/>	Unit	secondary	Neptune	Backup
<input checked="" type="checkbox"/>	<input type="radio"/>	DbService	DB1	Saturn	1
<input checked="" type="checkbox"/>	<input type="radio"/>	DbService	DB2	Neptune	2
<input type="checkbox"/>					
<input type="checkbox"/>					

Start Close

3. Select a startup option.

A number of options are available for creating new engine database tables when starting an engine (see [“Managing an Engine Database” on page 124](#)):

Startup Option	Description
warm	<p>No new database tables are created during the primary engine unit startup process. All information stored in the existing database will be recovered. This option cannot be used the first time an engine is started.</p> <p>Note: If you need to maintain data between iIS releases, or dump and restore data for other reasons, you can use the Dump/Restore utility. For details, see “Dumping and Restoring Data” on page 126.</p>
cold	<p>All new database tables (current state, registration, and history log) are created during the primary engine unit startup process. The cold option should always be used the first time an engine is started. If used in subsequent engine starts, however, the cold option will cause deletion of the existing engine database before creation of a new one—all current state information, registration, and history log data will be lost.</p>
newLog	<p>New history log database tables are created during the primary engine unit startup process. All history log information stored in an existing database will be lost.</p>
newState	<p>New current state database tables are created during the primary engine unit startup process. All current state information stored in an existing database will be lost.</p>
newState newLog	<p>New current state <i>and</i> history log database tables are created during the primary engine unit startup process. All current state and history log information stored in an existing database will be lost.</p>
newState newRegistration	<p>New current state <i>and</i> registration database tables are created during the primary engine unit startup process. All current state and registration information stored in an existing database will be lost.</p>

CAUTION Be careful when specifying startup options. All options other than “warm” can result in loss of data. You are asked to confirm a cold start.

4. Disable the engine components you do not want to start.

By default, all components specified in your configuration file are enabled.

5. Click Start.

The governor, engine units, and database services are started, in that order.

NOTE The engine might take some time to start. To check the status of the engine, use the Engine > Status command as explained in [“Monitoring Engines and Engine Components” on page 118](#). For a quick indication of status, check the status indicator in the Start Engine window (see indicator states in [“Main Viewing Panel” on page 69](#)).

If the primary engine unit remains offline, then the engine has not started successfully. A typical reason for startup failure is that the database service has failed to open a session with the engine database.

Ultimately, you may have to check engine component log files. The log file for each component is written in the FORTE_ROOT/log directory of the node on which the component is executing. This is not necessarily the node on which you are running iIS Console.

Once an engine component has come online, you can access the log file directly from within iIS Console, as described in [“Engine Log Files” on page 210](#).

➤ **To start individual engine components**

1. In the iIS Console main window, select the engine whose components you want to start.
2. Choose Engine > Start. The Start Engine window displays (see previous page).
3. Deselect the engine components you do not want to start.

Examine the state icon of any component you want to start to ensure it is not already online.

4. Click Start.

If an engine component is already running, you get an error message.

Reconfiguring an Engine

An engine's component partitioning and startup properties are stored in its engine configuration file, which is read at engine startup time. To reconfigure an engine you modify the configuration file and restart the engine. Some limited reconfiguration, such as modifying the logging of historical state information, can be performed dynamically: you do not have to restart the engine (see [“How to Dynamically Modify Database Logging”](#) on page 115).

Most reconfiguration involves modifying engine component partitioning to accommodate changing load on an engine or component failure. For example, if the load on your engine increases, you might have to start up additional database services, or if a primary engine unit fails (say, the server blows up), you might have to start up an additional engine unit to serve as backup to the failover primary.

In these cases, you must modify your engine configuration file, in the first case to include additional database services, and in the second case to put an engine unit on a different server node.

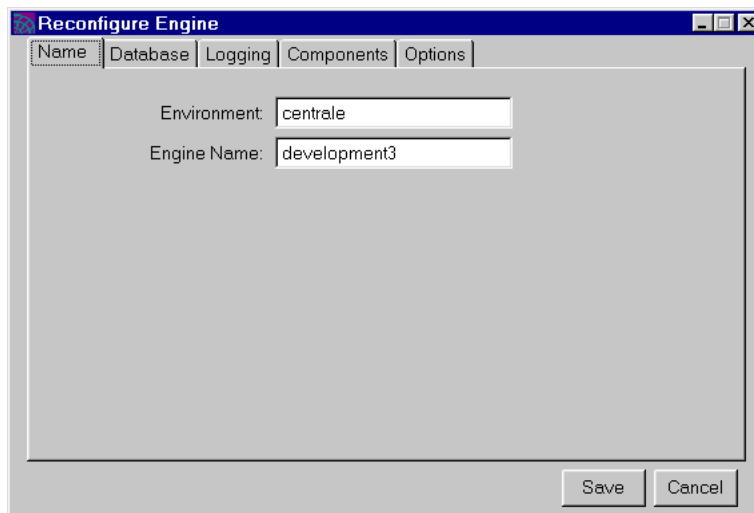
How to Reconfigure an Engine

You can statically reconfigure an engine whether or not it is currently ONLINE, either by editing the engine configuration file manually or by using the iIS Console as follows:

► **To reconfigure an engine**

1. Select an engine in the iIS Console main window.

2. Choose Engine > Reconfigure. The Reconfigure Engine window displays:



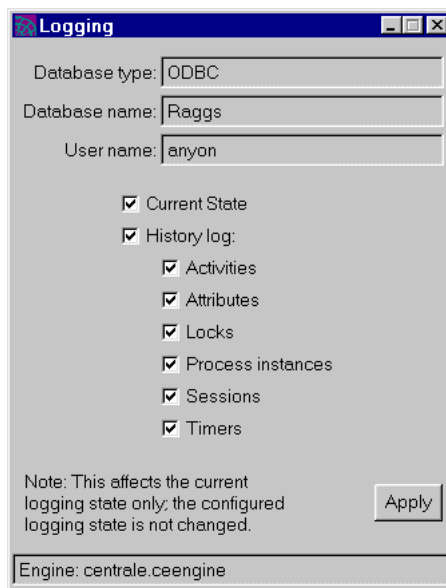
3. Select the appropriate tab to modify the property or properties you want to change.
4. When you have finished reconfiguring your engine, click Save.
The modified information is written to the engine configuration file.
5. Restart your engine for reconfiguration changes to take effect.

How to Dynamically Modify Database Logging

To change the logging configuration of your engine, normally to reduce or increase the amount of historical state information being logged to your engine database, you do not have to restart the engine.

► **To modify database logging for an engine**

1. Select an engine in the iIS Console main window.
2. Choose Engine > Logging. The Logging window displays:



3. Enable (or disable) the database tables to enable (or disable) logging.
4. Click Apply.

The changes you make are implemented immediately; however, they are not written to your engine configuration file. To make changes that will be saved to the engine configuration file, see [“How to Reconfigure an Engine”](#) on page 113.

How to Tune Process Execution

The performance of an engine depends on available system resources, such as memory. For example, if engine memory resources become overtaxed, engine performance can degrade, and an engine can even fail.

It is hard to know in advance the memory resources required under production loads. Memory requirements depend upon the number of sessions being supported and the number and complexity of concurrently executing process instances—the number of activities, timers, process attributes, and so forth. Hence, to prevent failure, it is necessary to monitor the performance of the engine under production loads, and to reconfigure it when performance begins to degrade.

A number of configuration options are available for tuning process execution, as described in [Step 13 on page 102](#):

- Maximum Number of Sessions
- Maximum Number of Processes
- Memory-resident Process Limit
- Swap-out Interval

For example, you can use the first two options to limit the number of concurrent sessions or process instances that the engine supports. While this can keep process execution within resource limits, it also limits the load that an engine can support.

A more sophisticated approach is to use the engine's advanced process management capabilities. These capabilities allow the engine to swap out dormant process instances in order to reuse the memory space they occupy. When these process instances are subsequently needed, they are swapped back into memory, replacing other process instances that are not currently being used.

In other words, only the most recently needed process instances are kept resident in memory. By swapping process instances out of and back into memory, the engine can support process execution loads that would otherwise exceed available memory resources and cause the engine to fail. Of course, there is overhead involved in this kind of process execution management, so in some cases, performance can be impacted.

The engine's mechanism for managing process execution depends on the values of two engine configuration options: the Memory-resident Process Limit and the Swap-out Interval. As process execution proceeds, the engine checks that the number of process instances resident in memory does not exceed the Memory-resident Process Limit. If it does, the engine swaps out the least recently

used process instances (and all their associated activities, timers, and process attributes), until it brings the number of memory-resident process instances back within the limit. The engine performs these checks periodically, as specified by the value of the Swap-out Interval.

When a swapped-out process instance is subsequently needed, for example, to change the state of an activity, to handle an expired timer, to evaluate an assignment role, and so forth, the engine finds the process instance in the engine database's current state tables and swaps it back into memory.

In general, you want the value of the Swap-out Interval to be short enough so that, under heavy load conditions, the number of process instances in memory cannot grow to a point where the available memory resources are exceeded. However, if you make the interval too short, the engine is incurring excessive overhead, swapping more process instances out of and into memory than is needed. Similarly, the value of the Memory-resident Process Limit must be set in accordance with the memory required by executing process instances (which depends on their complexity), the available memory resources, and the value of the Swap-out Interval.

Despite the difficulty of determining the optimum values of each of these configuration options, the combination provides great flexibility in tuning an engine to process the heaviest loads, with the greatest performance, and with the least risk of failure.

NOTE The evaluation of role-based assignment rules—as compared to assignment rules that involve process attributes or linked activities—does not require that the corresponding process instance be resident in memory. Hence, much less swapping is generally involved in executing process instances that employ only role-based assignment rules.

Monitoring and Changing Engine States

After you have started an engine, you should verify that it has started successfully and monitor it periodically. In some situations, you may want to dynamically change the state of engine components.

This section starts by describing how to monitor engines and engine components with the iIS Console, followed by details on how to change engine states. For information on how to perform these operations using Conductor Script, see [“Managing iIS Process Engines with Conductor Script” on page 235](#).

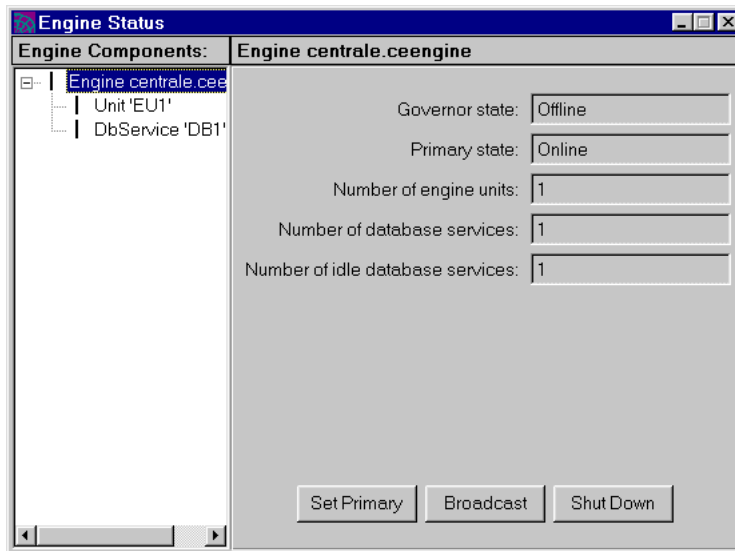
Monitoring Engines and Engine Components

You can get status information about the engine as a whole as well as about each of the individual engine component in the iIS Console (see [Chapter 3, “The iIS Console”](#)). iIS Console uses iPlanet UDS system management agents to gather data from engine components. Each agent has a set of instruments that report relevant information.

Monitoring the Engine

Typically, you want to know which engine components are running.

- **To check the engine runtime configuration**
 1. Select a running engine in the iIS Console main window.
 2. Choose Engine > Status. The Engine Status window displays:



The window provides information about the runtime state of the engine and engine component partitions. Partitions that are running (even if not fully functional) are shown with a warm-up state icon. For more detail on the functional state of a component, you can select that component (governor, unit, or DBservice) in the list view on the left, and the appropriate status display will appear on the right.

NOTE The Broadcast button lets you send a message to all open sessions. The message can be picked up by client applications or application proxies that have opened sessions with the engine (see [“Sending and Broadcasting Messages to Sessions”](#) on page 177).

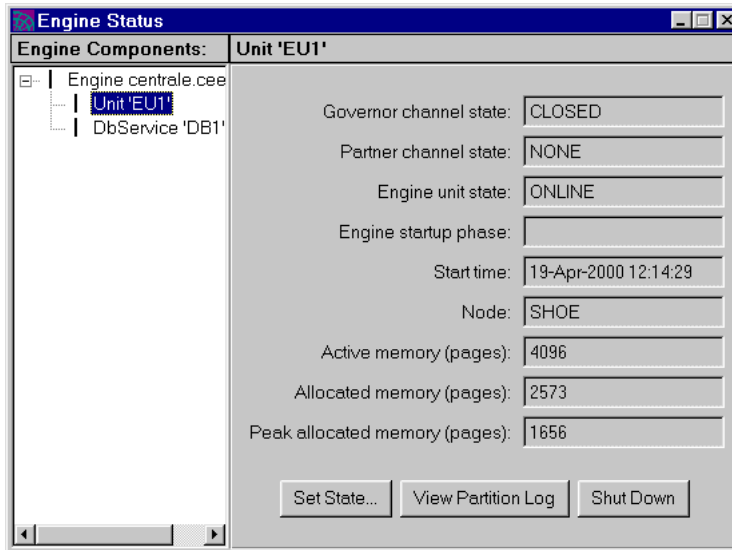
Monitoring Engine Components

You can get status information about individual engine components in the Engine Status window. Selecting an engine component in this window provides more information about the functional status of a component than is displayed in the Engine Status window if only the engine has been selected.

► **To monitor individual engine components**

1. Choose Engine > Status to display the Engine Status window.
2. In the list view on the left-hand side of the window, select the component to monitor: governor, engine unit, or database service.

The right side of the window provides information about the status of the engine component. For example, status information contained on a primary engine unit is shown below.



Engine unit In the case of an engine unit the following information is provided:

Property	Description
governor channel state	whether the communication channel to the governor is open or closed
partner channel state	whether the communication channel to the partner engine unit is open or closed
engine unit state	whether the engine is in ONLINE or STANDBY state or offline
engine unit startup phase	the startup phase of a primary engine unit (see “Primary Engine Unit Startup Phases” on page 108)
start time	the time at which the engine unit partition started
node	the node on which this engine unit is running
active memory	the size of the object memory heap currently used by the engine unit partition
allocated memory	the amount of active memory currently allocated to objects created by the engine unit partition

Property	Description
peak allocated memory	the amount of allocated memory remaining after the most recent memory reclamation (probably the best measure of active memory utilization)

Governor In the case of a governor the following information is provided:

Property	Description
governor state	the status of the governor's connections with engine units (IDLE, E1, BOTH, E2) as described in "Governor" on page 105
start time	the time at which the governor partition started up
node	the node on which this governor is running
active memory	the size of the object memory heap currently used by the engine unit partition
allocated memory	the amount of active memory currently allocated to objects created by the engine unit partition
peak allocated memory	the amount of allocated memory remaining after the most recent memory reclamation (probably the best measure of active memory utilization)

Database service In the case of a database service the following information is provided:

Property	Description
router channel state	whether the communication channel to the engine unit router is open or closed
database state	whether the database service is connected to the database or not
transactions processed	the number of completed database transactions since the start time of the database service
start time	the time at which the database service partition started up
node	the node on which this database service is running
active memory	the size of the object memory heap currently used by the engine unit partition

Property	Description
allocated memory	the amount of active memory currently allocated to objects created by the engine unit partition
peak allocated memory	the amount of allocated memory remaining after the most recent memory reclamation (probably the best measure of active memory utilization)

Changing Engine States

You can change the state of an engine, which usually involves changing the state of one or both engine units or shutting down one or more engine components. You can use the iIS Console to perform many of these tasks (see [Chapter 3, “The iIS Console”](#)). The iIS Console uses iPlanet UDS system management agents to control engine components.

Changing Engine Unit States

You might want to change the state of an engine unit for a number of reasons. For example, you might want to suspend all client sessions with a primary engine unit (change its state from ONLINE to STANDBY) so you can perform administrative functions on the engine database. Or you might want to place a primary engine unit in STANDBY state so you can make its partner primary. When an engine goes back to ONLINE from STANDBY, it recovers state information from the database.

- **To change the state of an engine unit**
 1. Select a running engine in the iIS Console main window.
 2. Choose Engine > Status. The Engine Status window displays.
 3. In the list view, select the engine unit whose state you want to change.
 4. Click the Set State button to display the Set Engine Unit State window.
 5. Click the radio button corresponding to the new engine unit state: ONLINE or STANDBY.
 6. Click the Set button.

- **To put the primary engine unit on STANDBY and the backup unit ONLINE**
 1. Select the running engine in the iIS Console main window.
 2. Choose Engine > Status. The Engine Status window displays.

3. Click the Set Primary button in the Engine Status window. The primary unit now becomes the backup unit, and the backup becomes the online unit.

About Recovering State Information

In a production system, placing a backup unit online requires recovering state information from the engine database. State information includes the state of every process instance, the state of every activity and timer within each process instance, the values of process attributes and their lock states, the state of routers and triggers for each activity instance, session activity lists, queue lists, and so on.

Accordingly, recovery of state information is not instantaneous. It takes an amount of time proportional to the amount of state information in the engine database, and inversely proportional to the speed of database access. The amount of state information available for a given process instance is determined by the process *recovery level* specified in the process definition or by the client application.

The recovery level is really a performance parameter that specifies how much state information about a process instance is stored persistently in the engine database. The less state information stored in the database, the higher the level of engine performance in executing that process instance, but the lower the level of recovery possible when a backup engine unit comes on line.

Recovery levels are specified on a per process basis (not per engine). There are three options.

None No state information is written to the engine database so none can be recovered—the process instance is therefore terminated when an engine fails and a backup unit comes on line.

Process only Minimal information is written to the engine database—only enough to recreate the process instance from its start. When an engine fails and a backup unit comes on line, all state information is lost and the process instance is recreated from start and executed anew.

Full recovery All state information for a process instance is written to the engine database. When an engine fails and a backup unit comes on line, state information is recovered as needed to proceed with process execution, that is, to evaluate assignment rules, change an activity state, and so on.

NOTE Writing of data to the engine database also depends on the logging properties of the engine configuration (see [“How to Configure a New Engine” on page 95](#)).

For more details on recovery levels, see the *iIS Process Development Guide* and the *iIS Process Client Programming Guide*.

Shutting Down Engine Components

You can shut down an individual component by clicking the Shutdown button in the component status display of the Engine Status window, or you can shut down all running engine components by clicking the Shutdown command in the engine status display.

Managing an Engine Database

The iIS engine database consists of three categories of tables:

Current State These tables maintain state information about all objects created in process execution so that the engine state can be recovered in case of engine failure. The tables contain current state information on all sessions, process instances, activities, process attributes, timers, and so on.

Registration These tables maintain state information about all library distributions and aliases (see [“About Aliases” on page 143](#)) registered with the engine. This information can be recovered in case of engine failure.

History Log These tables maintain historical information about the most important objects created in process execution (sessions, process instances, activities, timers, attributes, and attribute locks). This information can be used in managing process execution, analyzing historical trends, and reporting on business throughput. The log also includes information about engine startup, alarms, and registration that can be used for troubleshooting purposes.

Database Management Issues

There are a number of management issues to consider regarding your iIS engine database. These issues include growth of the database, failure of the database, and recovering your data. Each is discussed below.

Growth of the Database

As with any of the enterprise databases used by your workflow application, the engine database can grow in size. The current state tables grow and shrink in proportion to the number of sessions, process instances, activities, process attributes, and timers that exist at any one time. The registration tables are typically quite small, but grow in proportion to the number of distributions or aliases you register with the engine.

The history log, however, because it accumulates historical data, can grow quite large as time goes on, eventually reaching your database storage limit. You must therefore establish a set of procedures for monitoring and limiting the growth of the history log. The following suggestions might prove useful:

- Log only the minimum historical data that you will need for process execution analysis.

See [“How to Dynamically Modify Database Logging” on page 115](#) for history logging options.

- Maintain an archive history log database.

When rows in your history log tables are sufficiently aged, you can delete them from the engine database and transfer them to an archive database. For example, if your system does not make use of long-running process instances, you might want to periodically transfer all rows in your history log that correspond to process instances that terminated more than two months ago.

Failure of the Database

For any number of reasons, the engine database might fail. If the engine cannot access the database (that is, it cannot write to the database), then it raises an exception and transitions to STANDBY state in which it stops process execution. Engine exceptions are written to the primary engine unit’s log file and to the iIS Console Alarms window. See [Chapter 7, “Troubleshooting,”](#) for more information on exceptions.

Recovering Data

When starting an engine, you can choose to recover or lose any existing current state, registration, or history log database tables, as long as you are not starting the engine for the first time. The various startup options are described in [“How to Start an Engine” on page 110](#). When you recover the current state tables you also recover the registration tables.

Dumping and Restoring Data

The iIS Dump/Restore facility makes it possible to dump the data in an engine database to an ASCII file, and then, subsequently, to restore the data to the existing database or to a new engine database. It is used primarily to accommodate changes in database schema from one release of iIS to the next sequential release. However, you can also use it to back up a database, troubleshoot problems, or move data from one database system to another.

To dump or restore data for a particular engine database, the following conditions must be met:

- The iPlanet UDS Environment Manager must be running.
- The engine and database must have been configured, that is, the engine configuration file must exist.
- The engine must be shut down.

The Dump/Restore facility has two interfaces: a graphical user interface application and a command line utility. Both require information that you provide directly or by setting a number of environment variables—see [“Dump/Restore Environment Variables” on page 136.](#))

Dumping Database Tables

The following sections describe how to write the data in an engine database to an ASCII file. You can use either the Dump/Restore application or its command-line equivalent.

Using the Dump/Restore Application

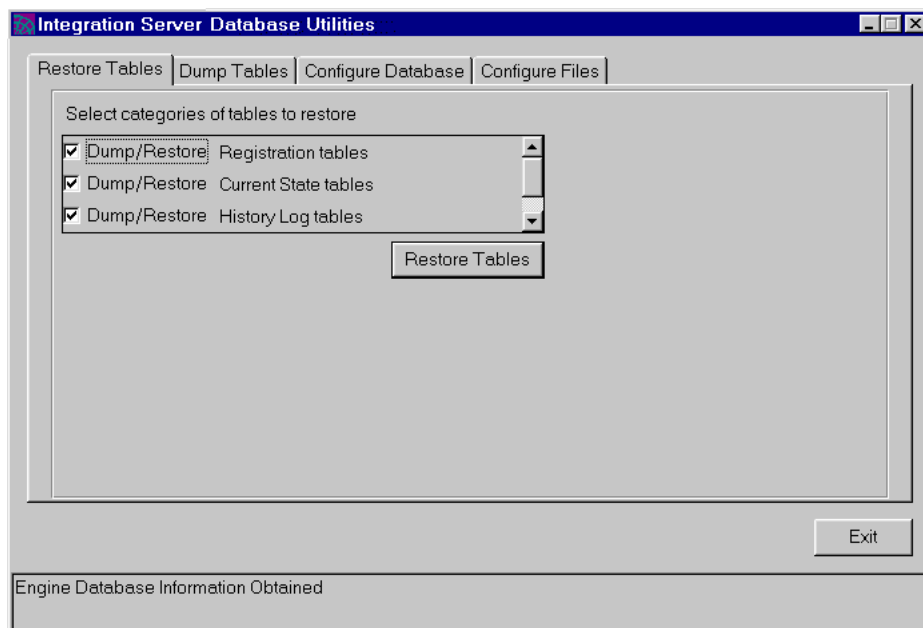
The following procedure shows how to use the Dump/Restore application.

➤ **To use the Dump/Restore application to dump database tables**

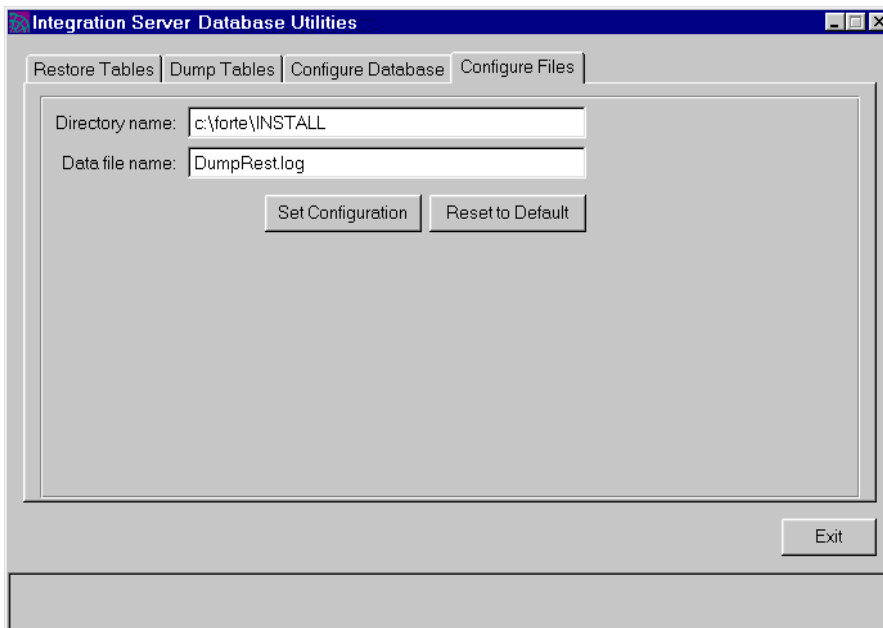
1. Start the Dump/Restore application.
 - on Windows or Windows NT: double-click the Dump/Restore icon.
 - on UNIX, OpenVMS, or Windows NT: enter the following:

```
ftexec -fnict -fcons
-fi bt:\$FORTE_ROOT\userapp\wfdrdump\c20\wfdrodu0
```

The iIS Dump/Restore window appears.



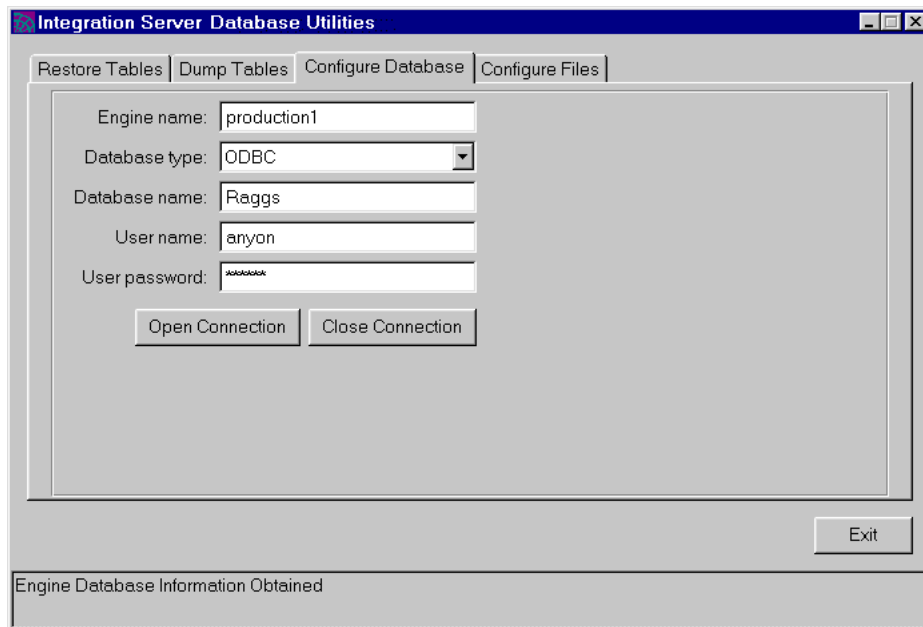
2. Configure a data file in which to dump the data.
 - a. Click the Configure Files tab.



- b. In the Directory name field, specify the directory into which the data file will be written.

The directory must already exist; the Dump/Restore application will not create a new directory.
 - c. In the Data file name field, enter the name of the data file.
 - d. Click the Set Configuration button.

3. Specify the engine database from which to dump the data.
 - a. Click the Configure Database tab.



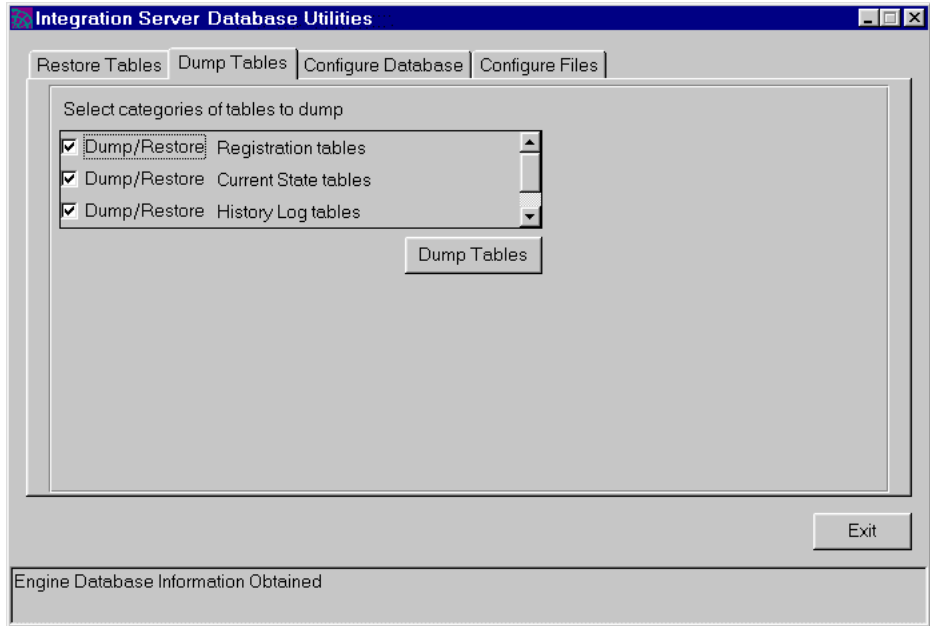
- b. In the Engine name field, enter the name of the engine whose data you want to dump.

When you exit the field, the Database type, Database name, User name, and User password fields will automatically be filled in from the engine configuration file.

- c. Click the Open Connection button.

The status bar at the bottom of the window displays a "Database Connected" message.

4. Dump the data.
 - a. Click the Dump Tables tab.



- b. Select the category of tables you want to dump.

By default, all categories are selected. Disable any category you do not want to dump.
- c. Click the Dump Tables button.
- d. Click OK in the Information dialog that is displayed.

While the dump is in progress, the status bar at the bottom of the window displays a "Dumping Tables" message. When the dump is complete, the status bar displays a "Completed Dumping Tables" message.

Using the Dump Command-line Utility

The Dump/Restore facility's `DrDump` command writes data in the engine database into an ASCII file.

Syntax for `drdump` command:

```
drdump -e engine_name [-d directory_name] [-v data_file_name]
[-t registration|state|history]
```

As in all iPlanet UDS command line specifications, if you use a name that includes a space, you should enclose the name in double quotation marks.

The following table describes the command line flags for the `DrDump` command:

Flag	Description
<code>-e engine_name</code>	The name of the iIS engine in the current environment. This flag is required.
<code>-d directory_name</code>	The directory in which to place the data files. The default value is the root directory.
<code>-v data_file_name</code>	The name of the data file. The default value is <code>dumprest.log</code> .
<code>-t registration state history</code>	The category of tables to dump. Valid values are REGISTRATION, STATE, HISTORY. The default is all categories. To specify a combination of two categories, use a colon (:) between categories. For example: <code>-t registration:state</code> Alternatively, you can enter the <code>-t</code> flag for each category. For example: <code>-e myEngine -t registration -t state</code>

If you have set the environment variables described in [“Dump/Restore Environment Variables” on page 136](#), then you do not need to supply values for the command line flags.

Restoring Database Tables

The following sections describe how to retrieve data which has been written to an ASCII file and restore it to an engine database. To restore data to an engine database, the engine must have been cold started. Cold starting an engine creates empty database tables that correspond to the engine database schema.

You can use either the Dump/Restore application or its command-line equivalent.

NOTE If you are upgrading to a new release and have customized your process engine database schema, be sure to read the following section, [“Preserving Engine Database Schema Customizations.”](#)

Preserving Engine Database Schema Customizations

If you are upgrading to a new release of iIS, the database schema for the process engine may have changed. If, in the previous release, you modified a process engine database schema to conform to your own specific requirements, as explained in “[Customizing Engine Database Schema](#)” on page 93, then you must perform the following procedure to preserve your modifications before restoring the database tables.

NOTE If you are not upgrading to a new release, or if the database schema did not change during the upgrade, then the following procedure is not necessary.

► **To preserve custom engine database schema changes when upgrading iIS**

1. Rename your current `engine.dbs` file as follows:
 - a. Navigate to `FORTE_ROOT/sysdata/conductr/cln/envname/engine.dbs`
envname is the name of your environment
engine is the name of your process engine
 - b. Rename `engine.dbs` to `engine.old`.
2. Start CConsole and reconfigure the process engine as follows:
 - a. In CConsole, select the process engine
 - b. Select Reconfigure, and then click Save
3. Compare the newly generated `engine.dbs` with `engine.old` created in [Step 1](#) of this procedure and make the following modifications:
Modify `engine.dbs`, based on the modifications from `engine..old`.
4. Cold start the iIS process engine you are upgrading.
Cold starting an engine creates empty database tables corresponding to the engine database schema.
5. Shut down the engine.

Using the Dump/Restore Application

The following procedure shows how to restore database tables that were previously written to an ASCII file, as explained in [“Dumping Database Tables” on page 126](#).

➤ **To use the Dump/Restore application to restore database tables**

1. Start the Dump/Restore application.

Follow the instructions under [“Dumping Database Tables” on page 126](#).

2. Configure a data file from which to retrieve the data.
 - a. Click on the Configure Files tab.
 - b. In the Directory Name field, specify the directory in which the data file is to be found.
 - c. In the Data File Name field, enter the name of the data file.
 - d. Click the Set Configuration button.

3. Specify the engine database in which to restore the data:

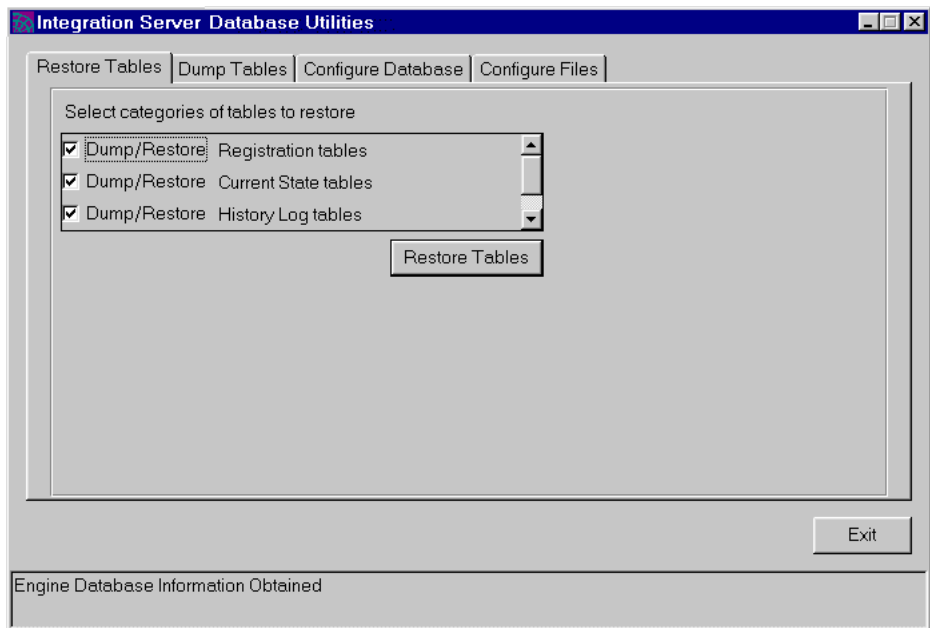
- a. Click the Configure Database tab.
- b. In the Engine name field, enter the name of the iIS engine whose data you want to restore.

When you leave the field, the Database type, Database name, User name, and User password fields will automatically be filled in from the engine configuration file.

- c. Click the Open Connection button.

The status bar at the bottom of the window will display a “Database Connected” message.

4. Restore the data.
 - a. Click the Restore Tables tab.



- b. Select the categories of tables you want to restore. By default, all categories are selected. Disable any category you do not want to restore.
 - c. Click the Restore Tables button.
 - d. Click OK in the Information dialog that is displayed.

When tables have been successfully restored, the status bar displays the "Completed Restoring Tables" message.

Using the Restore Command-line Utility

The Dump/Restore facility's `DrRestore` command retrieves data which has been dumped to an ASCII file and restores it into an engine database.

Syntax for `drrest` command:

```
drrest -e engine_name [-d directory_name] [-v data_file_name]
[-t registration|state|history]
```

As in all iPlanet UDS command line specifications, if you use a name that includes a space, you should enclose the name in double quotation marks.

The following table describes the command line flags for the `DrRestore` command:

Flag	Description
<code>-e engine_name</code>	The name of the iIS engine in the current environment. This flag is required.
<code>-d directory_name</code>	The directory in which to place the dump files. The default value is the root directory.
<code>-v data_file_name</code>	The name of the data file. The default value is <code>dumprest.log</code> .
<code>-t registration state history</code>	The category of tables to dump. Valid values are <code>REGISTRATION</code> , <code>STATE</code> , <code>HISTORY</code> . The default is all categories. To specify a combination of two categories, use a colon (<code>:</code>) between categories. For example: <code>-t registration:state</code> Alternatively, you can enter the <code>-t</code> flag for each category. For example: <code>-e myEngine -t registration -t state</code>

If you have set the environment variables described in [“Dump/Restore Environment Variables” on page 136](#), then you do not need to supply values for the command line flags.

Dump/Restore Environment Variables

iIS provides a set of environment variables for use with the Dump/Restore facility. You can set these variables to provide values for both the Dump/Restore application and the command-line utility.

Variable	Description
WFDR_ENGINE	The name of the iIS engine.
WFDR_DIRECTORY	The directory in which to place the dump files. The default value is the directory in which the application resides.
WFDR_DATAFILE	The default name for the data file. The default value is dumprest.log
WFDR_TABLETYPES	The category of tables stored in the database. Valid values are REGISTRATION, STATE, HISTORY. To specify more than one category of tables, separate the categories with a colon. For example: WFDR_TABLETYPE=registration:state

Managing Registrations

Registration is the procedure by which programmatic information about processes, users, and so on, created in the process development workshops, is made available to a running iIS process engine, so it can be dynamically loaded and used in process execution. The programmatic information usually consists of programs (shared libraries) created in the iIS process development workshops, but can also be logical references to executing processes, called aliases.

This chapter describes how to manage iIS process engine registrations. It covers the following topics:

- overview of registration concepts
- how to register iIS library distributions
- how to register an alias
- how to perform application upgrades

About Registration

The iIS process engine's principal function is to execute process definitions created in the iIS process development workshops. A process definition is a program that specifies how the engine coordinates the various client applications and resources that perform the work needed to complete a business process.

Most businesses are not static, but rather are dynamic: they want to create new business processes or modify existing ones. The organizational structure might also change, impacting how work gets assigned. To provide the flexibility businesses require, the engine can dynamically load and execute new and revised process definitions, assignment rules, and other programmatic information.

Registration is the procedure by which process logic created in the iIS process development workshops is made available to a running engine (see “**What Does Registration Do?**” below).

The entities that get registered with an iIS engine (described in detail in the *iIS Process Development Guide*) include the following.

user profiles Defined in the User Profile Workshop, a user profile is logic that specifies a user information template. The engine uses this template to build individual user profiles for each engine session. These user profiles are evaluated by assignment rules in determining which users can perform activities.

validations Defined in the Validation Workshop, a validation contains a ValidateUser method that the engine uses to validate a user (say, against an organization database) before opening a session with the engine.

assignment rule dictionaries Developed in the Assignment Rule Workshop, an assignment rule dictionary is a set of rules used by the engine to assign activities to users or other resources.

process definitions Developed in the Process Definition Workshop, a process definition is executed by the engine for each instance of the corresponding process. The engine manages and controls the process instance from creation through termination.

aliases Used in the Process Definition Workshop to specify subprocesses, an alias is a logical reference to an executing process that provides the flexibility to execute the process on any engine.

Registration capability is provided by the iIS process development workshops. These workshops let application system designers and process developers test their work by registering it with a test engine.

Registration capability is also provided by iIS process management tools (iIS Console and Conductor Script). These tools let system managers register process definitions, assignment rules, user profiles, validations, and aliases with production engines in production environments.

What Does Registration Do?

The entity actually registered when you register a process definition, assignment rule, user profile, or user validation is a library. (Aliases are not libraries and this section does not apply to them—see “[About Aliases](#)” on page 143.)

A library (often referred to as a shared library) is code that can be loaded into memory at runtime, and then referenced by any number of executing programs. A library distribution is the set of distribution files used to install one or more libraries on any particular node.

In the case of iIS engine registration, library distributions are installed on the nodes hosting iIS engine units so the libraries can be dynamically loaded and executed by the engine unit or units running on the node.

iIS library distributions are generated automatically when a developer selects the File > Distribute command in the Process Definition, Assignment Rule, User Profile, and Validation workshops. Code in the central development repository is extracted and made into library distribution files, which are then placed in a standard location on the central server node.

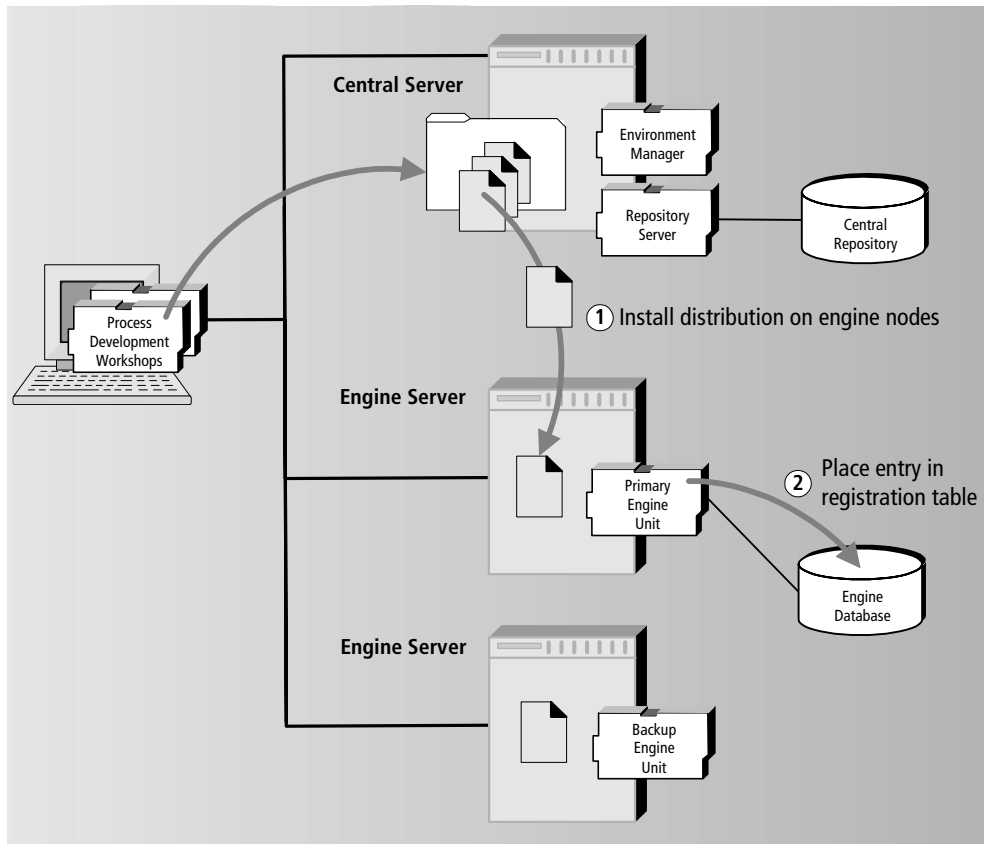
NOTE You can also generate library distributions using the `MakeConductorDistribution` Conductor Script command, if you have access to the central repository where development code resides. (iIS Console does not provide this capability.)

A special registration service (WFEnvAgent) running on the central server node keeps track of the iIS library distributions (process definitions, assignment rule dictionaries, user profiles, and validations) that have been made from the process development workshops or Conductor Script. Whenever an iIS library distribution is made, its compatibility level is augmented to distinguish it from any previously generated distributions.

After a library distribution has been made and it is located on the central server node, you, as a system manager, can use the iIS Console or Conductor Script to register the library with any number of engines. (iIS process developers can also register iIS library distributions for test purposes using a Register option of the File > Distribute command in the process development workshops.)

Registration consists of two steps, illustrated in [Figure 5-1](#), both transparent to the user performing the operation.

Figure 5-1 Registration Steps



- Installing library distribution files on the nodes hosting the target engine (more specifically, hosting the target engine units).

This step is performed by iPlanet UDS system management agents. The highest compatibility level library distribution residing on the central server node is installed on the nodes hosting both the primary and backup engine units. For details of how iPlanet UDS deploys library distributions see the *iPlanet UDS System Management Guide*.

- Placing an entry in the registration table of the target engine's database.

This step is performed by iIS engine unit agents and requires that the target engine be ONLINE.

If at some point you have to move an engine unit to a new node, you do not have to manually re-register (reinstall) all the libraries on the new node. When an engine unit first comes on line, if the registered libraries are not currently installed on the host node, iPlanet UDS automatically installs them. This operation, however, might take some time.

Registration in Production Environments

At a production site removed from a development environment, registration can only be performed using iIS process management tools. In a production environment it is more likely that registrations will be batched, rather than performed one at a time as in a development environment. It is also more likely that library distributions will be registered with a number of engines, rather than a single test engine.

Before performing registrations in such situations, however, you have to first transfer iIS library distributions to the production environment. Use the following steps:

► To transfer iIS library distributions to a production environment

1. Make sure that developers have generated all relevant iIS library distributions using either the File > Distribute command in the process development workshops or the `MakeConductorDistribution` Conductor Script command.
2. Copy the iIS library distributions located on the central server node in the development environment to your distribution medium.

The distributions are found in the following directory:

`FORTE_ROOT/appdist/environment_ID/distribution_ID`

where, `environment_ID` is the name of the active iPlanet UDS development environment. Copy the contents of the `distribution_ID` directory and the required `cl#` directory structure beneath it. For information on the structure of library distributions, see the *iPlanet UDS System Management Guide*.

3. Transfer the iIS library distributions from your distribution medium to the central server node in the production environment.

The distributions should be placed in the `FORTE_ROOT/appdist/environment_ID/` directory, where, `environment_ID` is the name of the active iPlanet UDS production environment.

4. Follow the instructions for registering iIS distributions in [“Registering iIS Distributions” on page 144](#).

What Does Unregistration Do?

Unregistering process definitions, assignment rules, user profiles, or a validation does not reverse the entire registration process, just the last step of the process: it removes the registration entry from an engine’s registration database table. It does not remove the shared library files installed on the engine server node, nor does it delete the library distribution from the environment repository (where they are placed in order to be installed on engine server nodes). You can delete a library distribution from the environment repository using the Conductor Script `Uninstall` command (see [“Uninstall” on page 321](#)).

Engine Registration Manager

Each engine has a Registration Manager that tracks library distributions registered with the engine. The manager ensures that all engine references to a library are to the most current registered version of the library.

For example, if you register a new version of a process definition with an engine, all subsequent instances of that process definition created by that engine are based on the new version. Process instances based on the older version, however, continue to execute to completion (process termination). When instances of the older version no longer exist, the engine registration manager automatically unregisters the old process definition.

When you register a new version of an assignment rule dictionary with an engine that has an older version already registered, the new versions of assignment rules are retroactively applied to all existing offered and queued activities. Offered activities are offered again to sessions based on the new rules, and access to activities in queues becomes governed by the new rules. The engine registration manager automatically unregisters the old versions of any assignment rules included in the new assignment rule dictionary.

Unregistering a library distribution removes the corresponding entry from the registration database table but does not delete the library files from the engine unit’s host server node.

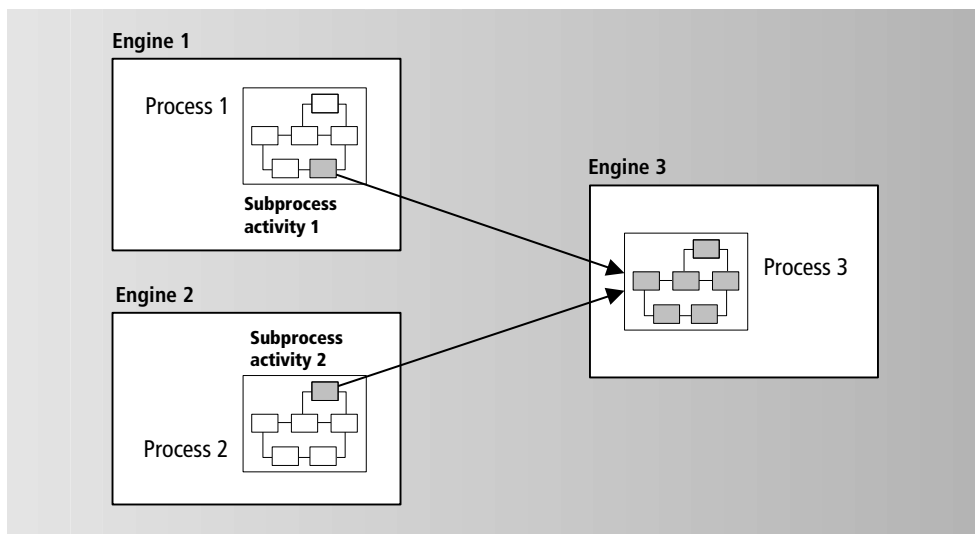
About Aliases

An alias is a reference to a process definition registered with an engine. It is used by developers when defining a Subprocess activity in a process definition so they do not have to hard code the process name and host engine represented by the Subprocess activity. Aliases provide the flexibility to move subprocess execution to different engines at runtime.

An alias is a logical name that is evaluated by the engine during process execution. It references a particular process definition executed by a particular engine. An alias must be registered with every engine executing a process definition containing the alias (that is, containing a Subprocess activity referencing the alias).

For example, suppose engine1 is executing a process1 that includes a Subprocess activity1, as shown in [Figure 5-2](#).

Figure 5-2 Subprocess Activity References



The Subprocess activity1 is actually a process3 that executes on engine 3. Suppose also that engine2 is executing a process2 that includes a Subprocess activity2. The Subprocess activity2 is also process3, which executes on engine3.

If process1 and process2 hard-code the reference to process3 on engine3, then process3 could not be moved to another engine without modifying both process1 and process2 in the Process Definition Workshop, creating the respective library distributions, and re-registering the process definitions with both engine1 and engine2. However, if process1 and process2 use an alias to reference process3 on engine3, then process3 could be moved to another engine by simply registering a new version of the alias with both engine1 and engine2.

To register an alias, specify the alias name (same as the process name) and the name of the engine executing the process definition. See [“Registering Aliases” on page 148](#).

Registering iIS Distributions

Process definitions, assignment rule dictionaries, user profiles, and validations are all registered using the same procedure. Each of these types of library distributions, however, has unique registration characteristics, as described briefly below:

User Profile A user profile must be registered with every engine. The user profile is used along with the validation to authenticate users who attempt to open sessions with the engine. The user profile is also used by assignment rules in determining who has permission to perform activities. You can register more than one user profile with each engine, but this is normally done only for upgrade purposes. Each user profile registered with an engine must be uniquely named. (An upgraded validation and upgraded assignment rule dictionary, consistent with the upgraded user profile, must also be registered—see [“Performing Application Upgrades” on page 153](#)). You cannot register an upgraded user profile of the same name as a previously registered user profile.

Validation One, and only one, validation can be registered with an engine, which allows the engine to open sessions with client applications. Unlike process definitions, assignment rule dictionaries, and user profiles, there can be only one validation registered with an engine at any time. Subsequent registrations replace the existing validation with a new one (implicitly unregistering the old one).

Assignment Rule Dictionary If a process definition references an assignment rule, the dictionary that contains that assignment rule must be registered. Generally, designers group assignment rules into dictionaries containing more than one assignment rule and then generate library distributions containing all the assignment rules in the dictionary. When you register an assignment rule dictionary with an engine, you have registered all the assignment rules contained in the dictionary. The engine uses only the most recently registered assignment rule, retroactively applying it to all existing activities.

Process Definition A process definition must be registered with every engine that can execute that process definition. Upgraded process definitions can be registered without unregistering the old versions: the old process definitions are used for existing process instances and the upgraded versions are used for all process instances started after the new definition is registered.

Registration Order

The distributions you register with an engine can be registered in any order with one exception: the user profile supplier of a validation or assignment rule dictionary must be registered before the validation or assignment rule dictionary is registered.

Since design elements are created before the process definitions that use them, it is normal for a user profile, validation, and assignment rule dictionary to be registered in that order, before process definitions are registered. However, in the course of development, assignment rules can change, or new ones may be developed, and these assignment rules can be registered at any time (if their user profile supplier does not change).

Performing Registrations

To register an iIS distribution, the library distribution must have been made and must reside on the central server node. The nodes hosting the target engine (or engines) must be online, and the primary engine unit of each target engine must be in ONLINE state.

As you perform registrations, iIS writes a log of registration operations. The log file is stored on the central server node in the following location:

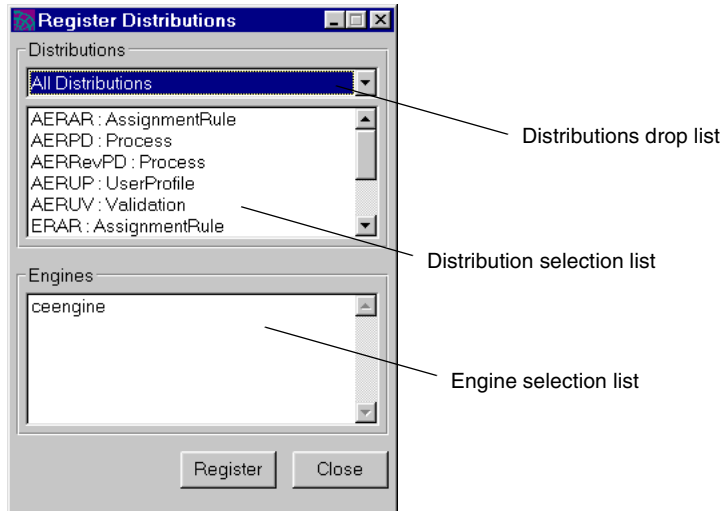
`FORTE_ROOT/sysdata/conductr/cl0/environment_name/engine_name.log`

NOTE After registration of an iIS distribution has taken place, subsequent relocation of an engine unit to another node is not a problem—library distributions required by the engine unit are installed automatically (from the central server node) when the engine unit first comes on line.

You can register one or more distributions with one or more engines, all at the same time.

► **To register one or more distributions using the iIS Console**

1. Choose Environment > Registrations> New. The Register Distributions window displays:



2. To filter the list of distributions available for registration, select Process Definitions, Assignment Rule Dictionaries, User Profiles, or Validation from the Distributions drop list.
3. In the Distribution selection list, select the distribution (or distributions) to register.
4. In the Engine selection list, select the engine (or engines) with which you want to register the specified distributions. All ONLINE engines are displayed.
5. Click OK to perform the registrations.

Upgrading Registrations

Sometimes you must register a new version (upgrade a registration) of one of these library distributions. How you upgrade a registration depends on the upgrade registration characteristics of the distribution type, as summarized in the following table:

Distribution Type	Registration Characteristics
User profile	Cannot register upgraded versions of the same name unless you first unregister the older version, cold start the engine, or rename the upgraded version.
Validation	Can register upgraded versions of the same name. Engine supports only the most recent version—automatically unregisters earlier version.
Assignment rule dictionary	Can register upgraded versions of the same name. Engine supports only the most recent version, retroactively applying it to all existing activities.
Process definition	Can register upgraded versions of the same name. Engine supports multiple versions. Engine unregisters older versions no longer being used.

Unregistering iIS Distributions

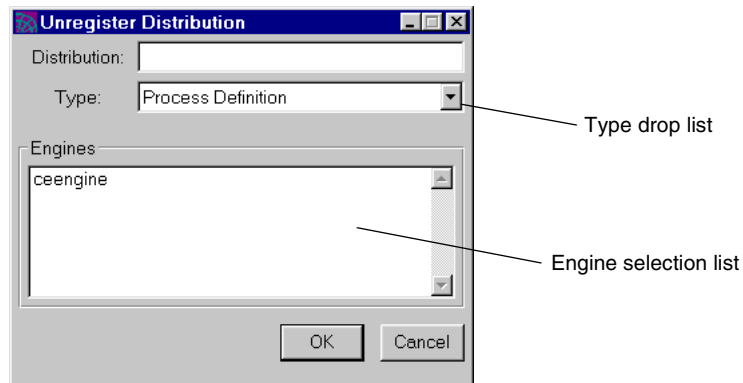
At times you might want to unregister a distribution from one or more engines, normally because those engines no longer use the process definitions they contain. (You might also have to unregister a distribution because for some reason you want to re-register it.) Cold starting an engine unregisters all registrations; however, if you want to be more selective, use the procedure described in this section.

NOTE You cannot explicitly unregister a validation. This is because the engine maintains only one validation—registering a new validation implicitly unregisters the previous one.

You can unregister only one registered distribution at a time.

► **To unregister a process definition, assignment rule dictionary, or user profile**

1. Choose Environment > Registration > Unregister. The Unregister Distribution window displays:



2. In the Distribution field, enter the name of the process definition, assignment rule dictionary, or user profile to unregister.
3. From the Type drop list, select the distribution type.
4. In the Engine selection list, select the engine (or engines) from which you want to unregister the specified process definition, assignment rule dictionary, or user profile. All running engines are displayed.

NOTE You cannot unregister a validation. The validation is deleted when a new validation is registered.

5. Click OK to perform the unregister operation.

Registering Aliases

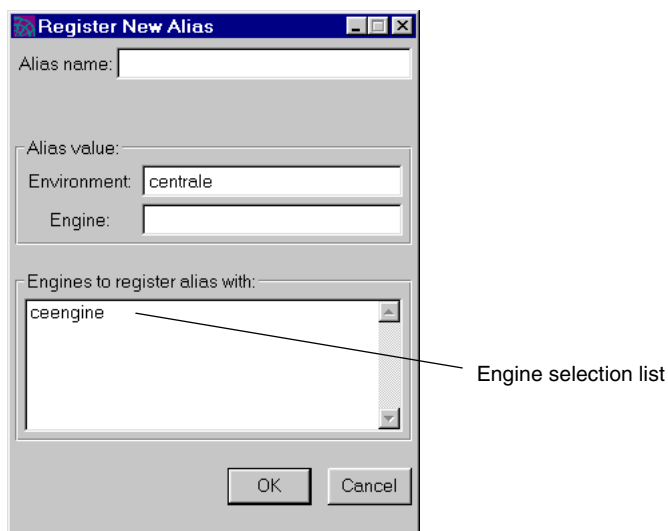
An alias must be registered with every engine executing a process definition that references the alias. Generally, developers provide you with a list of aliases used in their process definitions, and the subprocesses these aliases specify. You then have to register these aliases with the appropriate engines, as explained below.

Unlike process definitions, assignment rule dictionaries, user profiles, and the validation, aliases do not involve library distributions. In registering an alias, you are simply making an entry in an engine's registration table. You can register only one alias at a time, but you can register it with more than one engine. The registration process is the same for both new and modified versions of an alias.

To register an alias, the nodes hosting the target engine (or engines) must be online, and the primary engine unit of each target engine must be in the ONLINE state.

► **To register an alias using the iIS Console**

1. Choose Environment > Aliases > New. The Register Alias window displays:



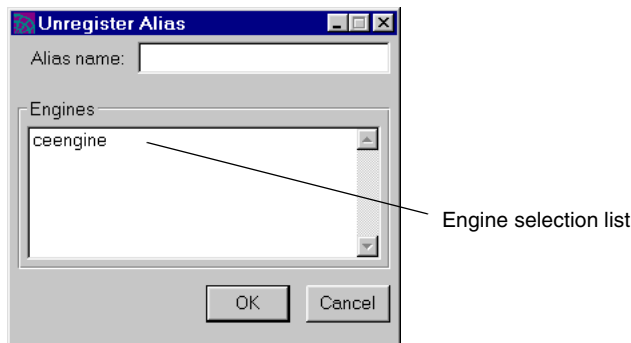
2. In the Alias name field, enter the name of the alias, as supplied to you by developers.
The alias name is the same as the process name of the subprocess.
3. In the Engine name field, enter the name of the engine the alias is referencing.
This is the name of the engine with which the specified process definition is registered and on which it will be executed. (The engine where the actual subprocess resides.)
4. Select the engines with which to register the specified alias.
5. Click OK.

Unregistering Aliases

At times you might want to unregister an alias from one or more engines, normally because those engines are no longer using it. You can unregister only one alias at a time.

► **To unregister an alias**

1. Choose Environment > Aliases > Unregister. The Unregister Alias window displays:



2. In the Alias name field, enter the name of the alias you want to unregister.
3. Select the engines from which to unregister the specified alias.
4. Click OK.

Viewing Registrations for an Engine

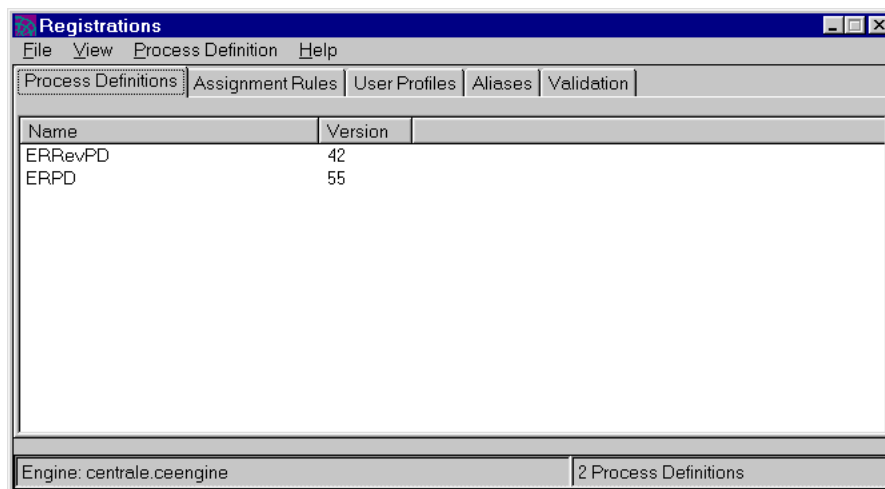
The registration procedures described in the previous sections let you perform registration and unregistration of iIS distributions on an environment-wide basis, that is, for multiple engines within an environment. However, iIS Console lets you view registration status only for individual engines.

For any engine, you can view the user profiles, validation, assignment rule dictionaries, process definitions, and aliases that are registered with that engine. You can also unregister any distribution registered with the engine. In the case of process definitions, you can monitor all instances of a registered process definition.

► **To view the registrations for a given engine**

1. Select a running engine in the iIS Console main window.
2. Choose Monitor > Registrations, or choose Registrations from the popup menu.

The Registrations window displays:



3. Click a tab to view a particular type of registered distribution.
4. Check that the window's refresh options are appropriately set.

Unregistering a Distribution

From the Registrations window, you can unregister any distribution currently registered with an engine, except a validation. As with the environment-wide registration facilities, you can only unregister one distribution at a time.

► **To unregister a registered distribution**

1. Select a running engine in the iIS Console main window.
2. Choose Monitor > Registrations, or choose Registrations from the popup menu.

The Registrations window displays.

3. Click a tab to open it to view a particular type of registered distribution.

The menu bar changes depending on the type of registration you are viewing. For example, if you are viewing assignment rule distributions, you see the File, View, and Assignment Rule menus. In the next step, the menu that changes is referred to as the *TabName* menu.

4. Select the item to unregister and choose *TabName* > Unregister.

NOTE You cannot unregister a validation. A validation is automatically deleted when a new validation is registered.

Monitoring Instances of a Registered Process Definition

From the Registrations window, you can monitor instances of any registered process definition.

NOTE You can also monitor instances of a registered process definition using the Monitor > Processes Resident command of iIS Console. However in this approach you get a list of *all* executing process instances in an engine, so you have to filter the resulting list by process name. In the procedure below you start with a list of only those process instances that correspond to a given registered process definition.

► To monitor instances of a registered process definition

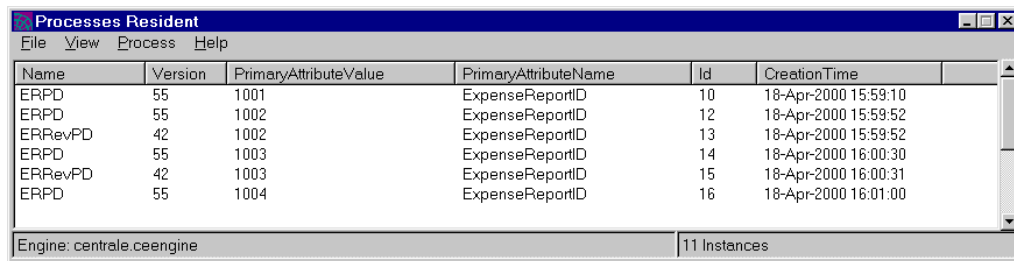
1. Select a running engine whose process instances you want to monitor in the iIS Console main window.
2. Choose Monitor > Registrations.

The Registrations window displays with the Process Definitions tab open.

3. Select the registered process definition whose instances you want to monitor.

4. Right-click and choose Process Definition > Open from the popup menu.

The Processes Resident window displays:



The screenshot shows a window titled "Processes Resident" with a menu bar (File, View, Process, Help) and a table of process instances. The table has columns for Name, Version, PrimaryAttributeValue, PrimaryAttributeName, Id, and CreationTime. The data rows are as follows:

Name	Version	PrimaryAttributeValue	PrimaryAttributeName	Id	CreationTime
ERPD	55	1001	ExpenseReportID	10	18-Apr-2000 15:59:10
ERPD	55	1002	ExpenseReportID	12	18-Apr-2000 15:59:52
ERRevPD	42	1002	ExpenseReportID	13	18-Apr-2000 15:59:52
ERPD	55	1003	ExpenseReportID	14	18-Apr-2000 16:00:30
ERRevPD	42	1003	ExpenseReportID	15	18-Apr-2000 16:00:31
ERPD	55	1004	ExpenseReportID	16	18-Apr-2000 16:01:00

At the bottom of the window, it displays "Engine: centrale.ceengine" and "11 Instances".

By selecting a process instance in the window and opening it (double-click or use the mouse popup menu), you can get the process instance's activity list, timer list, process attribute list, and so on. From these lists, in turn, you can get information about the state of any activity, timer, or process attribute.

Performing Application Upgrades

If the process logic of a business process application changes, you might need to upgrade process definitions, assignment rule dictionaries, user profiles, and validations—and even client applications or application proxies—depending on the type of change involved.

While properly specifying the upgrade of an iIS enterprise application is the responsibility of an application system designer, it is helpful for a system manager to understand the issues involved.

Under normal conditions, upgrading consists of registering new or updated process definitions to accommodate changes in the business process. This type of upgrade is straightforward.

In some situations, the new or updated process definitions might be based on new or modified assignment rules. In that case a new or modified assignment rule dictionary might have to be registered as well. (In some cases, you only have to register new or modified assignment rule dictionaries because the process definitions do not change.)

However, sometimes an upgrade involves a new or modified extended user profile. If this is the case, then the user profile, the validation and the assignment rule dictionaries that depend upon it, and client applications that reference it might all need to be upgraded. This type of upgrade has a broader impact, especially if you cannot shut down your engine to perform the upgrade.

Monolithic Upgrades

In a monolithic upgrade, your developers create an upgraded user profile and validation, upgraded assignment rule dictionaries, and upgraded client applications that are not compatible with the earlier versions. You must shut down your business process application to perform the upgrade.

► To perform a monolithic upgrade

1. All client applications and application proxies must close their sessions with the engine and shut down.

2. Unregister current assignment rule dictionaries and the user profile.

The validation does not need to be unregistered (in fact, it cannot be). Do not unregister process definitions.

3. Deploy each upgraded client application to all nodes supporting that client application.

4. Register the upgraded user profile, validation, and assignment rule dictionaries.

5. Register upgraded process definitions, if an upgrade is necessary.

6. Users can then start their upgraded client applications, open sessions with the engine, and resume work.

Rolling Upgrades

Rolling upgrades must be used in production situations where work cannot be interrupted and client applications cannot be upgraded monolithically. In a rolling upgrade, the changeover from an earlier version to an upgraded version takes place gradually. It requires that the engine simultaneously support client applications based on the earlier user profile as well as on the updated user profile.

To support differing versions of a user profile, designers create upgraded versions of the validation and the assignment rules that can accommodate both the new and old user profiles. Client application developers can then create new client applications based on the new user profile.

► To perform a rolling upgrade

1. Register the upgraded user profile, validation, and assignment rule dictionaries.
2. Deploy each upgraded client application to nodes supporting that client application.
3. When they choose to upgrade, users close sessions with the engine, shut down their old client applications, start up their new, upgraded client applications, open sessions with the engine, and resume work.

Managing Process Execution

The main function of an iIS process engine is to coordinate the work done by a variety of users (or application components) in performing the activities that comprise a business process. The engine does this by executing iIS process definitions that have been registered with the engine.

Managing process execution in an iIS system involves monitoring various aspects of process execution and making administrative adjustments if necessary.

This chapter first discusses the iIS process execution life cycle—from process creation to process termination—and then describes how to monitor and manage this process execution. The specific tasks covered include:

- monitoring and managing engine sessions
- monitoring and managing process execution
- providing historical analysis of process execution activity
- monitoring and managing two-phase commit transactions

Introduction

An iIS process engine creates instances of various process definitions and executes them from start to finish. During process execution, a succession of activities specified by the process definition are performed by client applications—or applications accessed through proxies—that have opened sessions with the engine. The engine manages and tracks these activities to their final completion, ensuring that they are performed in proper sequence.

This section discusses two topics basic to understanding process execution: engine sessions and activity states. More detailed information about process execution is provided in [“Process Execution” on page 165](#).

Engine Sessions

An iIS process engine generally coordinates the work of a number of applications that perform various activities that comprise a business process. To manage this process flow, the engine needs to maintain sessions with each of the applications involved. Consequently, the first order of business of a client application or application proxy is to establish a session with any engines managing activities to be performed by the corresponding application.

NOTE In the remainder of this chapter—unless explicitly stated—the term “client application” applies to client applications developed using iIS process client APIs, and also to application proxies used to integrate existing applications through backbone system capabilities provided by iIS. The application proxy interacts with an engine in the same way a client application does.

When a client application attempts to open a session with an engine, the engine must verify the authenticity of the application or user. This validation is performed by the engine using logon information provided by the application or user and validation code in the validation registered with the engine. This validation code normally compares information provided by the user with information stored in an organization database.

The session is the mechanism for all communication between a client application and the engine. It is used by the engine to offer activities in a process to different applications or users and to notify them about changes in the status of these activities. It is used by the client application to accept activities to perform and notify the engine when work on an activity is complete.

A session can have the following states:

Session state	Description
ACTIVE	A session to which the engine can assign activities and post events.
SUSPENDED	A session previously active, but now dormant. This state can result from a lost connection between client application and engine, or from explicit action by a system manager or client application user. The engine cannot assign new activities to a suspended session or post events to it.

Session state	Description
RECONNECTION_IN_PROGRESS	During engine recovery, a previously active session that is waiting to be restored to active during the engine recovery.

An ACTIVE or SUSPENDED session can be terminated by an application or user, or by a system manager. A RECONNECTION_IN_PROGRESS session can be suspended or terminated by a system manager. A session that has been terminated is deleted from the engine and the engine's current state database table.

Disrupted Sessions

A session is a two-way communication between client application and engine. Both the client application and the engine maintain an object representing the session. The state of the engine object determines the state of the session. The client application's session object uses code in the client API to restore the connection when a temporary disruption occurs. (A temporary disruption can be due to network interruptions or engine failure.)

The client session object tests the connection to the engine at regular intervals (refer to the iIS online help for the `WFSession.SetPingInterval`). When the client session object detects a disruption in the connection between the client application and the engine, it automatically attempts to restore the connection. If the client session object is unsuccessful in restoring the connection within a specified *auto reconnect timeout* period, it notifies the client application that the connection has been permanently lost. The auto reconnect timeout period is configured in a client application using the `WFSession` methods `SetRetry` and `SetPingInterval`.

Network failure The engine does not know about a network interruption until it touches an engine session object, for example, to post an event or assign an activity. If the engine finds the session disrupted, the engine suspends or terminates the session, depending on the value of the session's *disconnect action* property (set using the control parameter to `WFEngine.OpenSession` or `WFSession.SetControl`).

Engine failure When a primary engine unit fails, all active sessions are automatically set to RECONNECTION_IN_PROGRESS. During recovery, sessions remain in this state for a specified *timeout period* (set in the engine's configuration), waiting for the client sessions to reconnect. During this period, a system administrator can suspend or terminate these sessions.

If the client session reconnects to the engine during the timeout period, the sessions are restored depending on the value of the session's *reconnect action* property (set using the control parameter to `WFEEngine.OpenSession` or `WFSession.SetControl`). If the client session fails to reconnect, the sessions are suspended or terminated, according to the specification in the engine configuration.

If failover occurs (the partner engine unit comes ONLINE) within the auto reconnect timeout period, the client attempts to restore the sessions according to the value of the session's reconnect action property.

If the reconnect action property is set to `RECONNECT_ALLOWED`, then the session is restored to ACTIVE state. If the reconnect action property is set to `RECONNECT_PROHIBITED`, the session is not restored.

Session disconnect and reconnect parameters are summarized in the table below.

Session Property	Value	Description
disconnect action	SUSPEND	Engine suspends session if it finds a disruption in the connection between it and the client application.
	TERMINATE	Engine terminates session if it finds a disruption in the connection between it and the client application.
reconnect action	RECONNECT_ALLOWED	Allows client API code or a client application to restore a suspended session to ACTIVE state.
	RECONNECT_PROHIBITED	Prohibits client API code or a client application from restoring a suspended session to ACTIVE state. A new session must be created to replace the suspended session.

When the client API auto-reconnect mechanism attempts to restore a suspended session to an ACTIVE state, it passes the user name, password, and session name to the engine that was used in establishing the original session. (The client API stores this information.) The engine uses this information to confirm which suspended session should be restored to ACTIVE and whether the reconnection is allowed.

If a connection is permanently lost, the client API code notifies the client application that its connection has been permanently lost. The client application must explicitly reopen its session to restore it to ACTIVE state. Its ability to restore a suspended session to an ACTIVE state depends—just as with the auto-reconnect mechanism—on the value of the session’s reconnectAction property set when the session was first opened.

Explicitly Suspended Sessions

When a system manager explicitly suspends one or more sessions or changes the state of a primary engine unit from ONLINE to STANDBY, all affected sessions are automatically suspended and their corresponding client applications notified through an event. If the engine unit (or its partner) is subsequently restored to ONLINE state, the sessions are not automatically restored to ACTIVE state. After sessions are explicitly suspended, the corresponding client applications must explicitly reopen their sessions or create new sessions, depending on the values of the reconnectAction property set when each session was originally opened.

The effect of suspending a session on any activities a client application is performing (that is, on ACTIVE activities) depends on the suspendAction property set by the client application when it accepts the activity. For more information, see the description of the WFAActivity method StartActivity in the *iIS Process Client Programming Guide*.

Explicitly Terminated Sessions

When a system manager explicitly terminates one or more sessions, all such sessions are automatically terminated and their corresponding client applications are notified through an event. Termination ends sessions and deletes them from the engine and the engine’s current state database table. After sessions are terminated, the corresponding client applications must explicitly open new sessions.

NOTE Terminating a session on any activities a client application is performing (that is, on ACTIVE activities) aborts the activities. If an activity does not have an OnAbort router to accommodate an ABORTED state (see “[Activity States](#)” below), the engine aborts the process instance. You should therefore undertake the termination of sessions with caution.

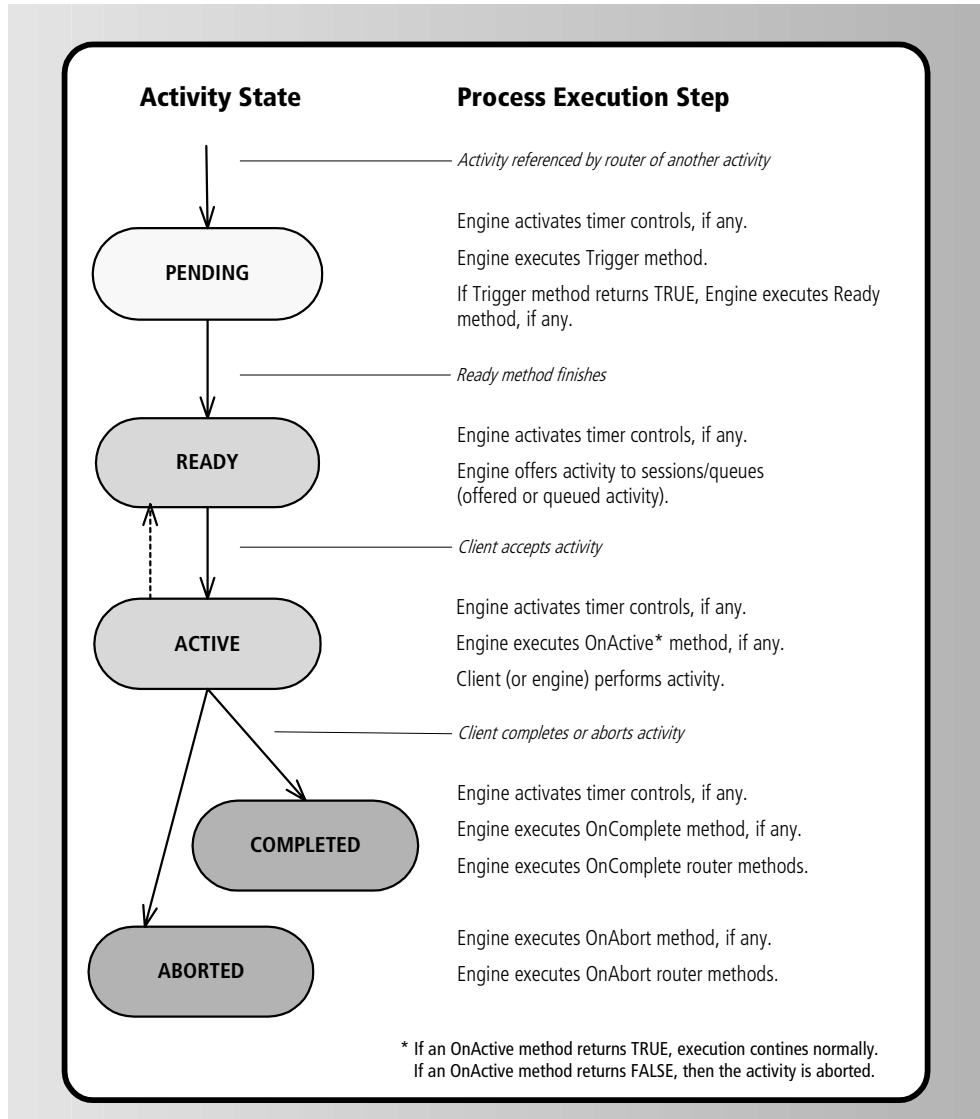
Activity States

During process execution the engine manages a succession of activities through a number of states, from creation to deletion. While there are a number of activity types, each with its own properties and behavior (see [“Activity Types” on page 165](#)), as a general rule, the engine takes each activity through a succession of states shown in the following table and illustrated in [Figure 6-1 on page 164](#). (For more detailed information on engine operations within each state and on transitions between states, see [“Activity Execution” on page 168](#).)

Activity state	Description
PENDING	An activity is normally created and placed in PENDING state when the router method of a completed (or aborted) activity or of an expired timer names the new activity. The activity remains in the pending state until all trigger conditions are met, at which time the engine performs any work specified by a Ready method in the process definition and then places the activity in READY state.
READY	When an activity is placed in READY state, it is made available to client sessions based on assignment rules specified for the activity in the process definition. Depending on the type of activity, the engine either offers it to each session whose user profile matches one or more of the activity’s assignment rules (offered activity), places the activity in a queue (queued activity), or places it directly in ACTIVE state (automatic activity). When offered to a session, the activity is placed on the session’s activity list (a list of offered activities maintained by the engine for each session) and is then available to the session’s client application. When placed in a queue, the activity is available to any session whose user profile matches one or more of the activity’s assignment rules.

Activity state	Description
ACTIVE	When a client application accepts an offered activity or takes it off a queue, it is placed in ACTIVE state. The engine performs any work specified by an OnActive method in the process definition and then provides the client application with process attribute data needed to perform the activity. The client application is responsible for performing the work associated with the activity. When the client determines that the work is complete, it informs the engine, which removes the activity from the session's activity list and places the activity in a COMPLETED state. When the client aborts the activity, it informs the engine, which removes the activity from the session's activity list and places the activity in an ABORTED state.
COMPLETED	The COMPLETED state signifies successful completion of an activity. The engine performs any additional work specified by the OnComplete method in the process definition, and then executes the activity's OnComplete router methods. Once any successor activities specified in the routers are created, the engine deletes the COMPLETED activity from memory.
ABORTED	The ABORTED state signifies unsuccessful completion of an activity. The engine performs any additional work specified by the OnAbort method in the process definition, and then executes the activity's OnAbort router methods. Once any successor activities specified in the routers are created, the engine deletes the ABORTED activity from memory. If the process definition does not provide OnAbort routers and the activity is aborted from ACTIVE state, the engine will abort the process instance.

Figure 6-1 Activity State Transitions—from Creation to Termination



Activity Types

A process definition can include a number of different activity types, each of which passes through the various activity states in a different fashion. The table below summarizes the different types of activities. For more information on activities, see the *iIS Process Development Guide*.

Activity type	Description
First	Specialized activity: the first activity in a process definition. Automatically placed in COMPLETED state, bypassing the PENDING, READY, and ACTIVE states.
Offered	An activity performed by client applications. Offered to sessions based on assignment rules and accepted by a client application from its work list. Passes through all states.
Queued	An activity performed by client applications. Placed on a queue and accepted by a client application. Passes through all states.
Subprocess	Represents a separate process executed by an engine. Passes directly from PENDING to ACTIVE state, bypassing the READY state.
Automatic	An activity performed directly by the engine. Not assigned to sessions. Passes through all states.
Last	Specialized activity: the last activity in a process definition. Passes directly from PENDING to a COMPLETED state, bypassing the intermediate states.

Process Execution

This section discusses how an engine executes an iIS process definition. Each process executes through a life cycle consisting of process instance creation, process instance execution, and process instance termination. Each of the six activity types—First, Offered, Queued, Subprocess, Automatic, and Last—plays a specific role in the overall life cycle.

Process instance creation Process creation involves the creation of a First activity.

Process instance execution Process execution involves creation of any number of activities, including one or more Offered activities (which are offered to and performed by client applications), Queued activities (which are placed on a queue and performed by client applications—but not, in this case, by applications integrated through proxies), Subprocess activities (which create an instance of another process), and Automatic activities (which are performed by the engine rather than client applications).

Process instance termination Process termination involves the completion of a Last activity.

The different activity types are discussed in the context of each stage of the execution life cycle, explained in more detail in the sections below.

Process Instance Creation

The engine creates a new instance of a process in three situations:

- A client application requests a new instance of a process

The request must provide the name of a process definition registered with the engine. The engine invokes the assignment rules for process creation to determine if the user making the request is authorized to create an instance of the process. If so, a First activity is created. A First activity passes automatically through the PENDING, READY, and ACTIVE states, and is placed directly into a COMPLETED state. Its OnComplete router methods, if any, are executed, and process execution proceeds.

- A Subprocess activity becomes ACTIVE

A Subprocess activity specifies a process and engine name. If the specified process is registered with the specified engine and all the required data can be located, the specified engine creates a new instance of the process. As in the previous case, a First activity is created, and process execution then proceeds.

- An engine recovers current state

During a failover scenario, or whenever a backup engine unit comes online, the engine needs to recover current state information from the engine database. In this situation, process instances that were active when the engine went off line are recreated and restored to their former states.

Process Instance Execution

Process execution consists of the sequential creation, execution, and termination of the activities specified in a process definition. When an activity is created, it remains in a PENDING state until its trigger conditions are fulfilled. In general, the activity then passes through READY and ACTIVE states, as shown in [Figure 6-1 on page 164](#), to a COMPLETED state. When the activity is completed, its router methods are executed and succeeding activities are created, continuing the process execution process.

The activity creation, execution, and termination stages are each discussed below.

Activity Creation

The engine creates an instance of an activity in the following situations:

- A new process is created—the engine creates a First activity, passes it through PENDING, READY, and ACTIVE states, and places it directly into a COMPLETED state.
- A completed activity's router points to an activity and the router method returns TRUE.
- An aborted activity's router points to an activity and the router method returns TRUE.
- An expired timer's router points to an activity and the router method returns TRUE.

In the last three cases, the engine creates an activity, and places in a PENDING state. However, if the target activity already exists in a PENDING state, a new instance is not created. (If the activity already exists in a READY or ACTIVE state, a new instance is created and placed in a PENDING state.)

The engine executes the trigger method of a PENDING activity in the following situations:

- The activity is first created
- The router method of a router pointing to the activity returns TRUE
- A process attribute for the process instance changes value

Activity Execution

As a general rule, activity execution consists of taking each activity through the series of states described in “[Activity States](#)” on page 162. In each state, the engine performs a number of operations before the activity transitions to the next state. As each state transition takes place, the new state is written into memory and logged in the state database tables. The engine activates any timer controls referenced by the new state and executes one or more activity methods that may be defined for that state in the process definition, as shown in [Figure 6-1](#) on page 164. When executing activity methods, the engine applies the process attribute locks specified for those methods in their respective attribute access lists, as defined in the process definition.

Despite these general rules, activity execution also depends to some degree on the type of activity being executed. The four types of activities that require work to be performed—Offered, Queued, Subprocess, and Automatic—are all executed somewhat differently by an iIS process engine. Execution of each activity type is discussed separately below.

Offered Activities

Transition to READY State When the trigger conditions are met, the engine performs any work specified in the Ready method (if one exists in the process definition) and places the activity in a READY state.

READY State Handling When an Offered activity is placed in a READY state, the engine offers the activity to sessions (that is, offers it to clients) based on the activity’s assignment rules.

During assignment, the engine searches for eligible sessions. The engine offers the activity to sessions based on the activity’s assignment rules (for example, to each session for which the session’s user profile matches at least one of the activity’s assignment rules). The engine continues assignment until all active sessions have been examined.

When an Offered activity is assigned to a session, it is placed on an activity list maintained by the engine for that session—the session activity list.

Session assignment takes place in any of the following situations:

- An offered activity is placed in a **READY** state for the first time.
- An offered activity is rolled back from an **ACTIVE** state to a **READY** state—can occur if a client application rolls the activity back to **READY** state (discards any work performed on the activity), if a system manager rolls the activity back to **READY** state in order to force it to be reassigned, or if the activity is being performed by a session that is suspended and the client has set the `suspendAction` property to **REMOVE** (that is, remove the activity from the session’s activity list if the session is suspended).
- A new session is opened (or reverts from **SUSPENDED** to **ACTIVE**)—the engine examines the list of **READY Offered** activities and assigns them to the newly active session based on each activity’s assignment rules.

Transition to ACTIVE State The engine places an Offered, **READY** activity in an **ACTIVE** state when a client application accepts the corresponding work item. The engine removes the activity from all sessions to which it is assigned except the one which accepts the corresponding work item.

The engine provides the client application with the associated application dictionary item (a reference to the application or service needed to perform the activity, a description of the work to be performed, and any required process attribute data). The engine ensures that any required process attributes are locked so that no other session can access them during performance of the activity.

ACTIVE State Handling The engine performs any work specified in the `OnActive` method (if one exists in the process definition).

The client application that accepted the activity is responsible for completing the work associated with the activity. When the client determines that the work is complete, it informs the engine, which removes the activity from the session’s activity list and places the activity in a **COMPLETED** state.

The client can abort work by requesting that the engine place the activity in an **ABORTED** state, or the client can also reset the activity to **READY** state (discarding any work performed on the activity, including changes in attribute values).

Queued Activities

Transition to READY State When the trigger conditions are met, the engine performs any work specified in the `Ready` method (if one exists in the process definition) and places the activity in **READY** state.

READY State Handling When a Queued activity is placed in a READY state, the activity is placed in a queue named after the queued activity and containing activities with the same name from multiple process instances. Client applications access the queue, and accept the highest priority activity on the queue. The priority ordering of activities in a queue can depend on the value of a queue prioritizing process attribute. If so, the queue is dynamically reordered whenever an activity is added to the queue and whenever the prioritizing process attribute changes value.

Transition to ACTIVE State The engine places a queued, READY activity in an ACTIVE state when a client application accepts the activity, then removes the activity from the top of the queue. When a client application accepts an activity, the engine evaluates the corresponding session against the activity's assignment rule to confirm that the client application is eligible to perform the activity.

The engine provides the client application with the associated application dictionary item (a reference to the application or service needed to perform the activity, a description of the work to be performed, and any required process attribute data). The engine ensures that any required process attributes are locked so that no other session can access it during performance of the activity.

ACTIVE State Handling The engine performs any work specified in the OnActive method (if one exists in the process definition).

The client application that accepted the activity is responsible for completing the work associated with the activity. When the client determines that the work is complete, it informs the engine, which removes the activity from the session's activity list and places the activity in a COMPLETED state.

The client can abort work by requesting that the engine place the activity in an ABORTED state, or the client can also reset the activity to a READY state (discarding any work performed on the activity, including changes in attribute values).

Subprocess Activities

Transition to READY State In this case the activity skips READY state and is placed directly in an ACTIVE state.

READY State Handling A subprocess activity passes directly from a READY to an ACTIVE state without being assigned to sessions.

Transition to ACTIVE State In the case of a subprocess activity, the engine places the activity directly in an ACTIVE state from a PENDING state. It skips a READY state altogether.

ACTIVE State Handling The engine performs any work specified in the `OnActive` method (if one exists in the process definition).

An active subprocess activity is handled differently depending on whether the subprocess activity is synchronous or asynchronous.

- Synchronous—waits for completion of the new process:
If the subprocess activity is *synchronous* and if the new process is successfully created, the subprocess activity remains in an `ACTIVE` state (in the parent process), awaiting completion of the subprocess. If the subprocess fails to complete for any reason, the subprocess activity is placed in an `ABORTED` state.
- Asynchronous—does *not* wait for completion of the new process:
If the subprocess activity is *asynchronous* and if the new process is successfully created, the subprocess activity is placed in a `COMPLETED` state (in the parent process). If the subprocess is not successfully created, the subprocess activity is placed in an `ABORTED` state.

Automatic Activities

Transition to READY State When the trigger conditions are met, the engine performs any work specified in the `Ready` method (if one exists in the process definition) and places the activity in a `READY` state.

READY State Handling An automatic activity passes directly from a `READY` to an `ACTIVE` state without being assigned to sessions.

Transition to ACTIVE State In the case of an automatic activity, the engine places the activity directly in an `ACTIVE` state from a `READY` state without assigning it to sessions because the work of the activity is invoked or performed by the engine rather than by a client application.

ACTIVE State Handling The engine performs the automatic activity's `OnActive` method when the activity is placed in an `ACTIVE` state. The work performed by the activity is invoked or coded in the `OnActive` method within the process definition. When the `OnActive` method is fully executed and returns `TRUE`, the activity is placed in a `COMPLETED` state.

Activity Termination

Activity termination occurs when the engine places an activity in either a `COMPLETED` state or an `ABORTED` state.

Completed Activities

When an engine places an activity in a COMPLETED state, the engine performs the following actions:

- executes the activity's OnComplete method, if one is specified in the process definition
- executes the activity's OnComplete router methods
- checks for a process stall condition—when no other activities are in an ACTIVE or READY state (that is, all are either COMPLETED or PENDING) and no timers are active—and aborts the process instance if a stall condition is detected
- deletes the COMPLETED activity from state information

In the case of a Last activity, however, instead of performing the above actions, the engine places the process directly in a COMPLETED state.

Aborted Activities

When an engine places an activity in an ABORTED state, the engine performs the following actions:

- executes the activity's OnAbort method, if one is specified in the process definition
- executes the activity's OnAbort router methods—if none are specified, the engine will, by default, abort the process instance
- checks for a process stall condition (no other activities are in an ACTIVE or READY state—that is, all are either COMPLETED or PENDING—and no timers are active) and aborts the process instance if a stall condition is detected
- deletes the ABORTED activity from state information

In the case of First and Last activities, however, the activities cannot reach an ABORTED state, since they proceed directly to a COMPLETED state.

Process Instance Termination

Process termination occurs when the engine places a process instance in either a COMPLETED state or an ABORTED state.

Completed Process Instances

An engine completes a process instance when the Last activity reaches a COMPLETED state.

Aborted Process Instances

An engine aborts a process instance in the following situations:

- the process instance is stalled—no activities are in an ACTIVE or READY state (that is, all are either COMPLETED or PENDING) and no timers are active.
- an ACTIVE activity (including a Subprocess activity) is placed in an ABORTED state and has no OnAbort router method specified.
- a system manager requests that a process be aborted.
- an exception is thrown on an activity method.

When a process instance is aborted, any uncompleted activities in the process are aborted, all process timers are stopped, and all process attribute updates are discarded.

Monitoring and Managing Engine Sessions

Since the engine is coordinating work being performed by client applications that have sessions with the engine, one dimension of system management concerns the number and state of such sessions.

This section describes how to perform the following tasks:

- Monitor the state of a session
- Manage one or more sessions

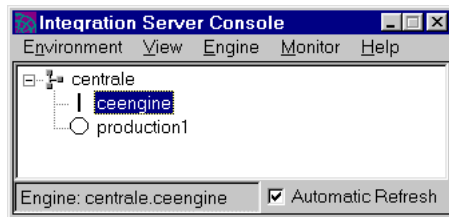
NOTE This section uses mouse popup menu commands to describe how to perform various operations. You can also use other methods of accessing commands, as described in [Chapter 3, “The iIS Console.”](#)

Monitoring the State of a Session

Often you want to know information about a particular session, such as its state, its creation time, the list of activities assigned to the session, or information about any such assigned activities that are in ACTIVE state.

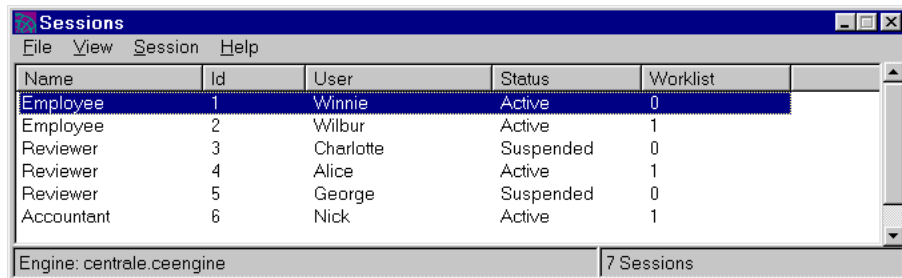
► **To obtain state information about a session**

1. Open the iIS Console and select the engine to monitor.



2. Choose Monitor > Sessions, or choose Sessions from the popup menu.

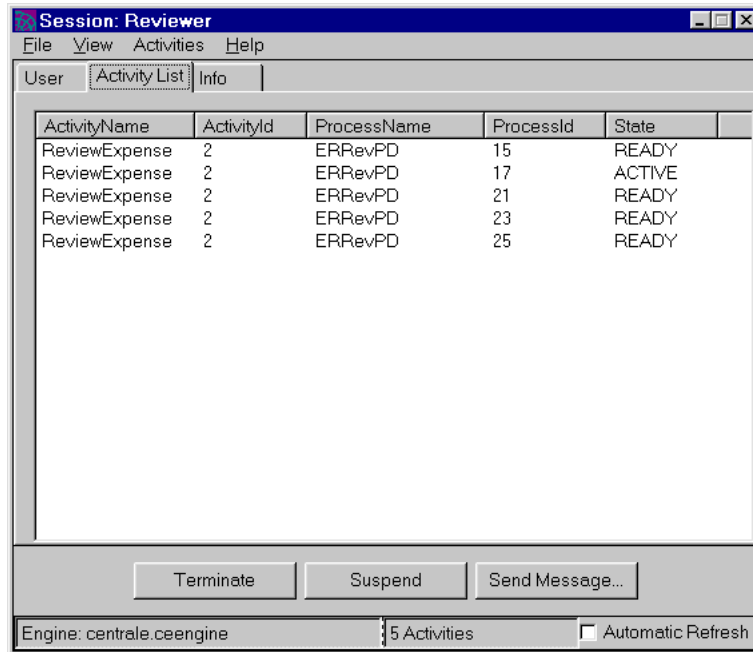
This opens the list of current sessions.



For each session in the list, the window displays an ID, the user who opened the session, the state of the session, and the number of items on the session's activity list.

- To get more information about a session, select it, and choose Session > Open from the popup menu.

An activity list for the session is displayed.



For each activity in the list, the dialog displays the activity ID, the process name and process instance ID in which it was created, and its current state.

NOTE To get more information about a given activity, such as how long the activity has been in the present state, and information about the corresponding process instance (process attributes, activities, timers, and so on), see [“Checking the Status of an Activity”](#) on page 185.

Managing Sessions

Session management consists of suspending or terminating sessions, or sending a message to one or more sessions (that is, the corresponding client applications).

Suspending or Terminating Sessions

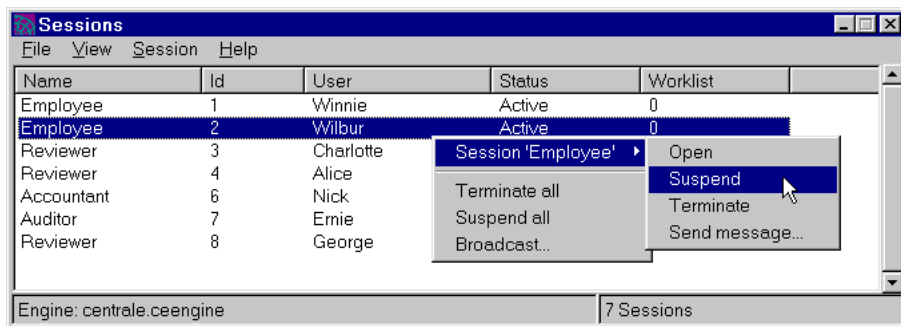
On occasion you might find it necessary to suspend or terminate one or more active sessions. Suspending a session posts a SessionSuspended event, and terminating the session posts a SessionTerminated event. These events allow the client application to handle the state change gracefully.

For the impact of suspending a session, see [“Explicitly Suspended Sessions” on page 161](#). For the impact of terminating a session, see [“Explicitly Terminated Sessions” on page 161](#).

NOTE A session that has in-progress two-phase commit operations can be suspended, but cannot be terminated. For more information, see [“Monitoring and Managing Two-Phase Commit” on page 199](#).

► To suspend a session

1. Choose Monitor > Sessions to display the Sessions window with the list of current sessions for the engine.
2. Select the session to suspend.



3. Choose Session 'Name' > Suspend from the popup menu.

You will be asked to confirm the suspension.

NOTE To suspend all sessions, choose Session > Suspend all.

► **To terminate a session**

1. Choose Monitor > Sessions to display the Sessions window with the list of current sessions for the engine.
2. Select the session to terminate.
3. Right-click and choose Session '*Name*' > Terminate from the popup menu.
You will be asked to confirm the termination.

NOTE To terminate all sessions, choose Session > Terminate all.

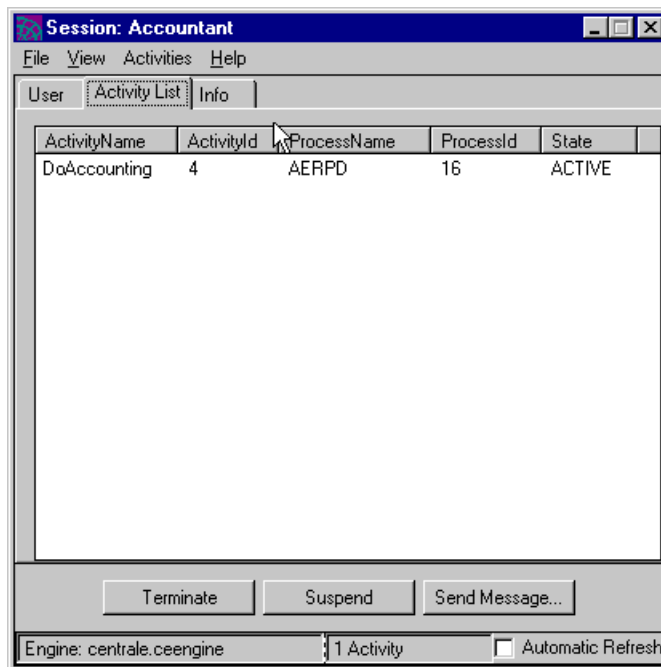
Sending and Broadcasting Messages to Sessions

You can send an informational message to any active session or broadcast the same message to all active sessions. A client application can watch for messages posted to its corresponding engine session.

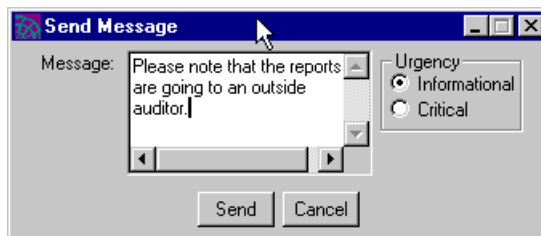
► **To send a message to an active session**

1. Choose Monitor > Sessions to open the list of current sessions for the engine.

- Open the session of interest.



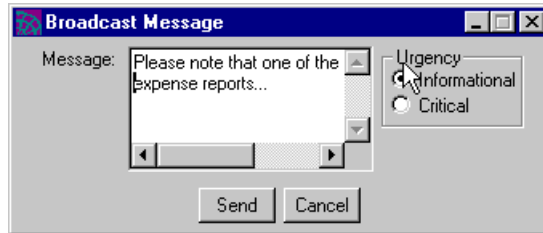
- Click Send Message. The Send Message window displays:



- Type the message in the Message field.
- Select the urgency: Informational or Critical.
- Click Send.

► **To broadcast a message to all sessions**

1. Choose Monitor > Sessions to open the list of current sessions.
2. Right-click and choose Session > Broadcast from the popup menu. The Broadcast Message window displays.

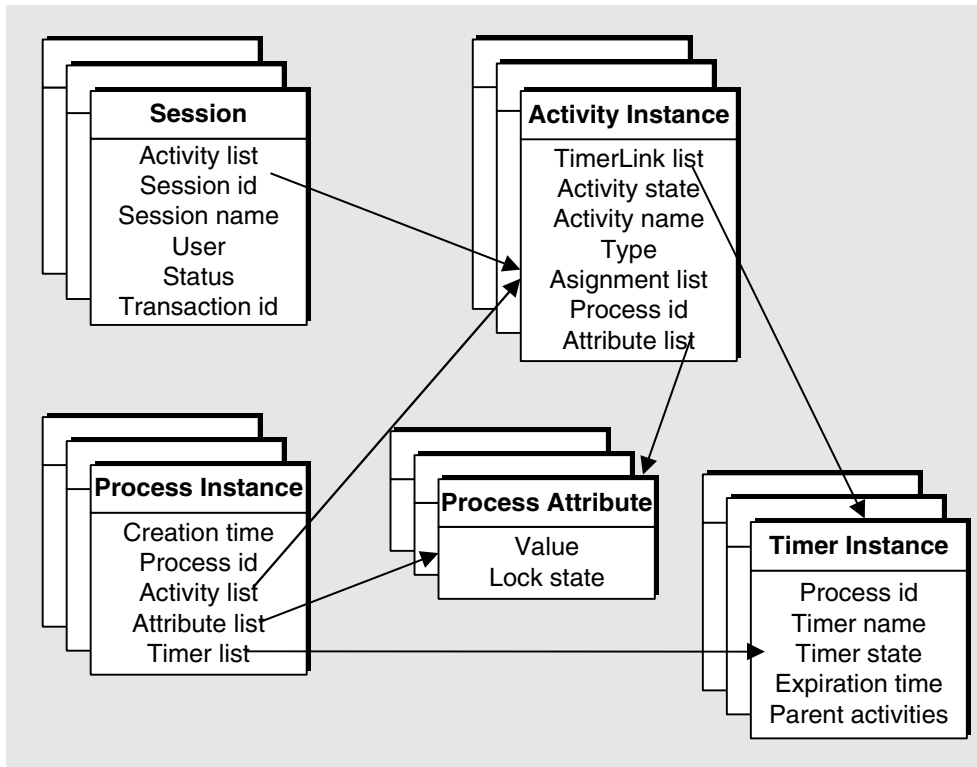


3. Type the message in the Message field.
4. Select the urgency: Informational or Critical.
5. Click Send.

Monitoring and Managing Process Execution

As described in [“Process Execution” on page 165](#), an iIS process engine executes process instances by creating a succession of activities, moving them through a number of states, checking timers, and locking, modifying, and unlocking process attributes.

The engine maintains information about all registered process definitions and the process instances corresponding to each. For each process instance, it maintains information about activities, timers, and process attributes (and locks placed on each process attribute). The information provided about these objects and their relationship to one another are illustrated in [Figure 6-2](#):

Figure 6-2 Process Execution Objects: Properties and Relationships

Using the iIS Console, you can display lists of these objects and filter them according to specific criteria. You can also open and view the properties (or state) of any specific process instance, activity instance, timer instance, or process attribute. In addition, There are a number of situations where you might want to intervene in process execution to resolve problems.

This section describes a number of common process execution monitoring and management tasks you might perform:

process instance Checking the execution status of a process instance and possibly aborting it.

activity Checking the status of an activity instance and possibly changing its state.

activity queue Checking the status of an activity queue and possibly reprioritizing an activity within the queue (for queued activities only).

timer Checking the status of a timer instance and resetting it or changing its state.

process attribute Checking the value and lock state of a process attribute and possibly changing its value and removing a lock.

In addition, you typically check for bottlenecks in process execution and diagnose their cause.

Managing Process Instances

This section describes how to check the execution status of a process instance and possibly abort it.

Checking the Status of a Process Instance

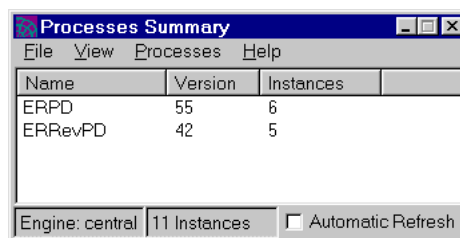
Suppose you want to find out the current execution status of a given process instance: which of its activities are in PENDING, READY, or ACTIVE state; how long they have been in those states; the values of important process attributes; and the state of process timers.

► To check the current execution status of a process instance

1. Select the engine executing the process in the iIS Console main window.

The engine must be running.

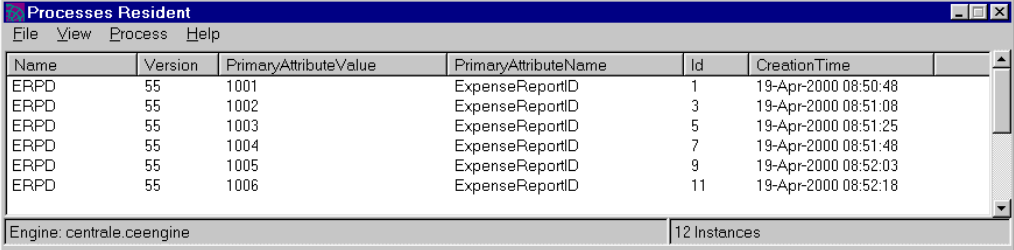
2. Choose Monitor > Processes Summary. The Processes Summary window displays, showing the list of process definitions, by name, for which there is at least one active process instance being executed by the engine.



For each process instance, the window displays the process name, the version of the process definition, and the number of process instances being executed.

3. Select the process name you want to view instances of and choose Processes > Open Instances of.

This opens the Processes Resident window, displaying the list of process instances being executed by the engine for the selected process name.



The screenshot shows a window titled "Processes Resident" with a menu bar (File, View, Process, Help). Below the menu bar is a table with the following data:

Name	Version	PrimaryAttributeValue	PrimaryAttributeName	Id	CreationTime
ERPD	55	1001	ExpenseReportID	1	19-Apr-2000 08:50:48
ERPD	55	1002	ExpenseReportID	3	19-Apr-2000 08:51:08
ERPD	55	1003	ExpenseReportID	5	19-Apr-2000 08:51:25
ERPD	55	1004	ExpenseReportID	7	19-Apr-2000 08:51:48
ERPD	55	1005	ExpenseReportID	9	19-Apr-2000 08:52:03
ERPD	55	1006	ExpenseReportID	11	19-Apr-2000 08:52:18

At the bottom of the window, it displays "Engine: centrale.ceengine" and "12 Instances".

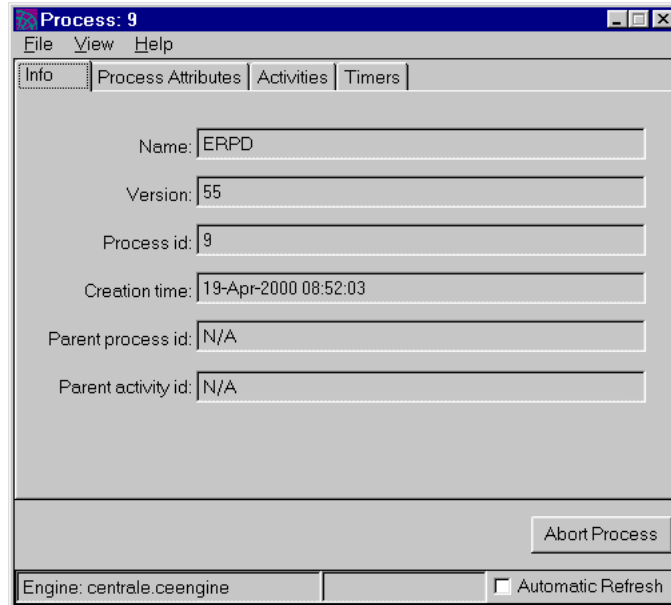
For each process instance the window displays the process name, the version of process definition, the value of the primary process attribute, the primary process attribute name, the process instance ID, and the process instance creation time.

NOTE You can open the Processes Resident window more directly by choosing Monitor > Processes Resident, however the instances displayed correspond to all process definitions rather than to a single one.

4. Search for a process instance by process attribute value, creator, or creation time.

You can use the Filter or Sort commands, available from the View menu, to help find specific process instances. For more information, refer to [“Filtering iIS Console Lists” on page 73](#) and [“Sorting iIS Console Lists” on page 76](#).

5. Select a process instance and choose Process > Open Instance.
The Process Instance property inspector is displayed.



By selecting the appropriate tab, you can get the process instance’s activity list, timer list, process attribute list, and other information. From these lists, in turn, you can get information about the state of any activity (see [“Checking the Status of an Activity” on page 185](#)), timer (see [“Checking the Status of a Timer” on page 191](#)), or process attribute (see [“Checking the Value and Lock State of a Process Attribute” on page 193](#)).

Aborting a Process Instance

If you want to stop execution of a process instance, you can abort that process instance. This aborts all uncompleted activities, stops all timers, and discards all process attribute updates.

► **To abort a process instance**

1. Open the property sheet of the process instance you want to abort.

See [Step 1](#) through [Step 5](#) under “[Checking the Status of a Process Instance](#)” on [page 181](#).

2. Click the Abort Process button.

NOTE You can abort all process instances displayed in a Processes Resident window by choosing Process > Abort All. If an error is encountered you are asked if you want to continue aborting the remaining processes or stop the operation.

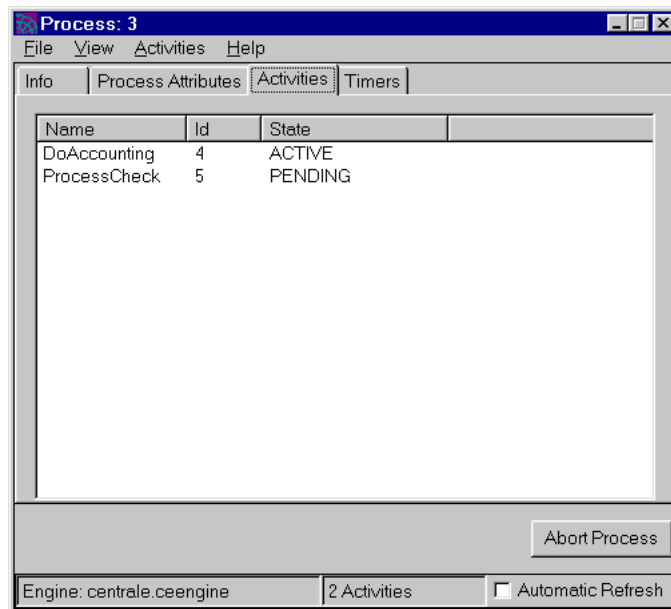
Managing Activity Instances

This section describes how to check the status of an activity and change its state.

Checking the Status of an Activity

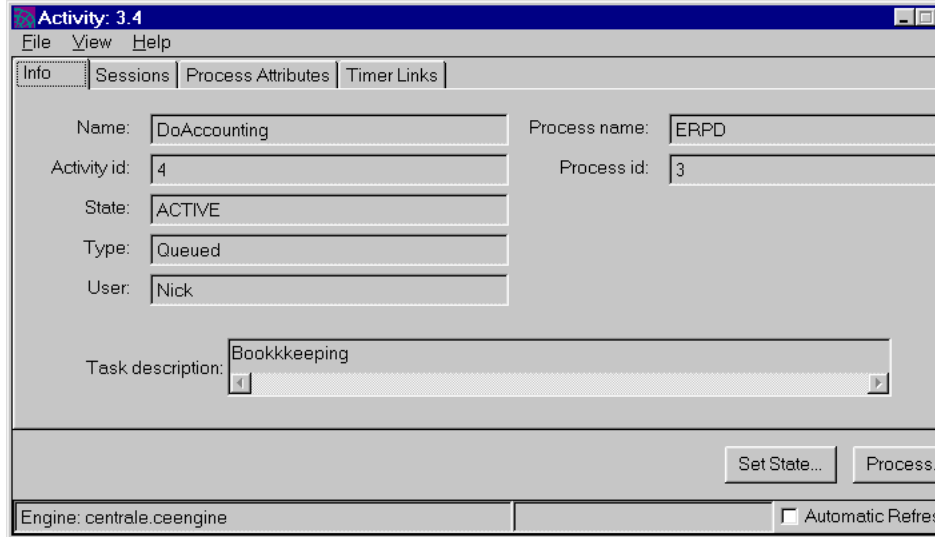
1. Select the activity in any window that contains an activity list:
 - o Process Instance property inspector
 - o Session property inspector
 - o Activities Resident window
 - o Activity Queues window

An example activity list (the Activities tab of a Process Instance property inspector) is shown below.



2. Choose Activities > Open.

The Activity property inspector displays:



The property inspector consists of a number of tabs: the Info tab displays the basic properties of the activity, and additional tabs provide information such as the sessions to which the activity is assigned, the current value of process attributes, and information about timers linked to the activity.

For example, if you click the Process Attributes tab, you get information about the current values of the process attributes for the process instance in which the activity was created.

NOTE You can click the Process button on the Activity property inspector to display the Process Instance property inspector.

Changing the State of an Activity

You can make the following state changes for an activity:

- from PENDING to READY

You might do this because one of the trigger conditions cannot be met for some reason, but you want the activity to go to the READY state anyway.

- from ACTIVE to READY

You might do this because the person working on an activity cannot complete it and you want it to be reoffered to other users. All process attributes are unlocked and rolled back to the values they had when the activity was first offered.

- from PENDING or READY to ABORTED

You might do this because some condition has arisen that precludes the completion of the activity and you want the OnAbort routing, if any, to take place. If there are no OnAbort routers or if all OnAbort routers return FALSE, the activity is simply aborted and any locked process attributes are unlocked.

- from ACTIVE to ABORTED

You might do this because some condition has arisen that precludes the completion of the activity and you want the OnAbort routing, if any, to take place. If there are no OnAbort routers or if all OnAbort routers return FALSE, not only is the activity itself aborted, but the whole process instance is aborted.

You cannot abort an ACTIVE synchronous subprocess activity—the activity is waiting for a subprocess to either complete or abort.

CAUTION Use caution when aborting an ACTIVE activity.

► **To change the state of an activity**

1. Open the property inspector for the activity whose state you want to change.
See [Step 1](#) and [Step 2](#) under “[Checking the Status of an Activity](#)” on page 185.
2. Click the Set State button.

The Set Activity State window displays, with the current state filled in.



3. From the New state drop list, select the new state of the activity.
4. Click the Set button.

Managing Activity Queues

This section describes how to check the status of an activity queue and how to reprioritize an activity within the queue. Activity queues only apply to queued activities.

Checking the Status of an Activity Queue

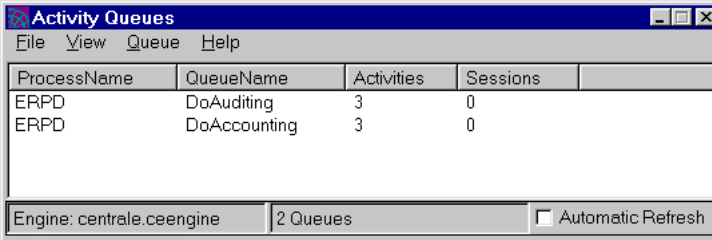
Occasionally you might need to check the queues that store queued activities until users are ready to work on them. You can display a list of activity queues and open a queue to display the contents of the queue. Doing so might tell you, for example, that a particular activity is not being performed because it remains at the bottom of a queue.

► To view the contents of an activity queue

1. Select the engine maintaining the activity queue in the iIS Console main window.

The engine must be running.

2. Choose Monitor > Activity Queues. The Activity Queues window displays, showing the list of queues in the engine:



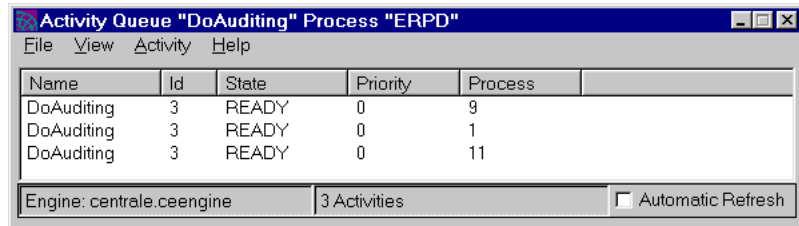
The screenshot shows a window titled "Activity Queues" with a menu bar (File, View, Queue, Help). The main area contains a table with the following data:

ProcessName	QueueName	Activities	Sessions
ERPD	DoAuditing	3	0
ERPD	DoAccounting	3	0

At the bottom of the window, there is a status bar showing "Engine: centrale.ceengine" and "2 Queues". There is also a checkbox labeled "Automatic Refresh" which is currently unchecked.

For each queue, the window displays the process name and queue name (which is the same as the name of the queued activity), the number of activities in the queue, and/or the number of sessions waiting for an activity to be placed in the queue.

3. Select a queue.
4. Choose Queue > Open. The Activity Queues window displays, showing the list of activities in the queue.



For each activity in the queue, the window displays the activity name, activity ID, activity state, prioritizing value (value of the queue prioritizing process attribute), and process ID.

If you want more information about an activity in the queue, you can open the activity property inspector by selecting the activity and choosing Activity > Open (see [“Checking the Status of an Activity”](#) on page 185). If you want to change the position of an activity in the queue, see [“Reprioritizing a Queued Activity,”](#) below.

Reprioritizing a Queued Activity

If a queued activity is not making it to the top of an activity queue because the value of its queue prioritizing process attribute always places it toward the bottom of its queue, you can change the value of the process attribute to reprioritize the activity in the queue. You must be careful when performing this operation, however, since other process logic might also depend on the value of the process attribute.

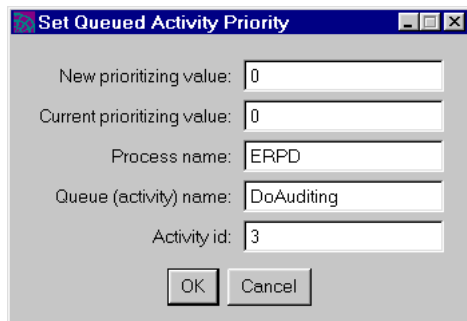
► To reprioritize a queued activity

1. Open the Activity Queue window for the queued activity.

See [Step 1](#) through [Step 4](#) under [“Checking the Status of an Activity Queue”](#) on page 188.

2. Select the activity in the activity list.

3. Choose Activity > Set Priority. The Set Queued Activity Priority window displays:



The screenshot shows a dialog box titled "Set Queued Activity Priority". It contains the following fields and values:

- New prioritizing value: 0
- Current prioritizing value: 0
- Process name: ERPD
- Queue (activity) name: DoAuditing
- Activity id: 3

Buttons: OK, Cancel

4. In the New prioritizing value, change the value of the queue prioritizing process attribute.
5. Click OK.

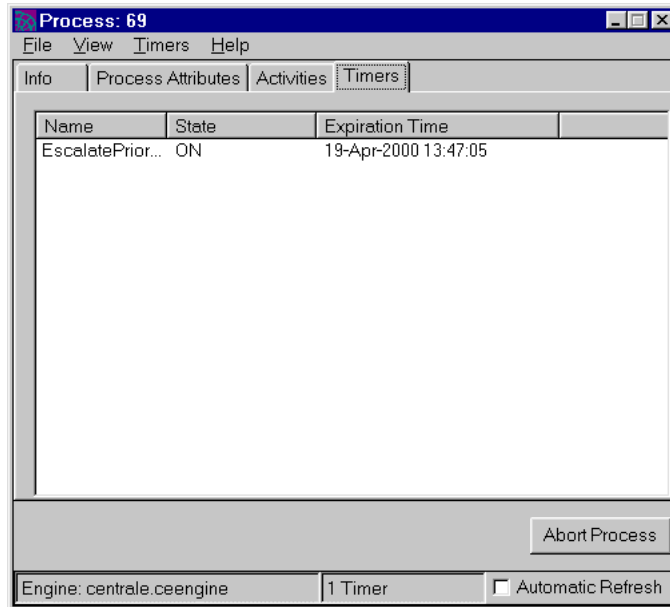
The activity will be placed in a new position in the activity queue, based on the new value of the queue prioritizing attribute.

Managing Timer Instances

This section describes how to check the status of a timer, reset it, or change its state.

Checking the Status of a Timer

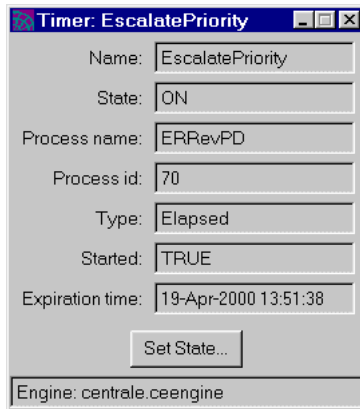
1. Open the Timers tab of a Process Instance property inspector, shown below.



For each timer, the list displays the timer name, timer state, and expiration time.

2. Select a timer.

3. Choose Timers > Open. The Timer property inspector displays, with the basic properties of the selected timer:



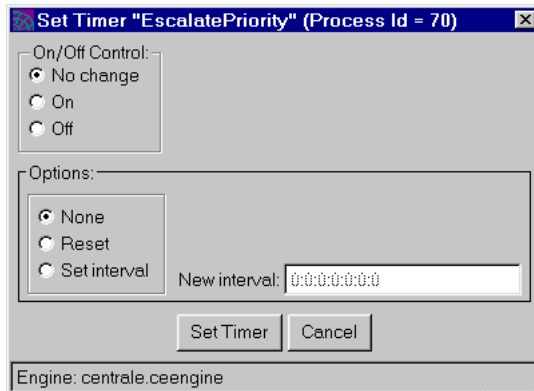
Changing the Timer State and Expiration Time

► **To change the state and expiration time of a timer**

1. Open the timer's property inspector.

See [Step 1](#) through [Step 3](#) under *“Checking the Status of a Timer”* on page 191.

2. Click the Set State button. The Set Timer window displays:



3. To change the state of the timer, click one of the radio buttons in the On/Off Control.
4. To change the expiration time, click one of the radio buttons in the Options box.
 - For an elapsed timer (screen shown above), Reset sets the original elapsed time interval and Set Interval lets you enter a new elapsed time interval in IntervalData format (years:months:days:hours:minutes:seconds:milliseconds).
 - For a deadline timer (screen not shown), Reset sets the deadline (expiration time) to its original value and Set Deadline lets you enter a new expiration time, in DateTimeData format (dd-nnn-yyyy hh:mm:ss, for example 03-Feb-1993 22:45:12).
5. Click the Set Timer button.

Managing Process Attributes

This section describes how to check the value of an attribute and its lock state and to change the attribute value or release a lock.

Checking the Value and Lock State of a Process Attribute

1. Select the attribute in any window that contains a process attribute list:
 - the Process Attributes tab of a Process Instance property inspector
 - the Process Attributes tab of an Activity property inspector

An example attribute list (the Process Attributes tab of a Process Instance property inspector) is shown below.

Name	Type	Value	Lock
ExpenseReport...	Integer	1002	None
Status	Integer	3	None
TotalAmount	Real	100	None
_CreatedBy	String	"Winnie"	None
_CreationTime	DateTime	19-Apr-2000 08:51:08	None
_Name	String	"ERP.D"	None
_ID	Integer	3	None

Engine: centrale.ceengine 7 Attributes Automatic Refresh

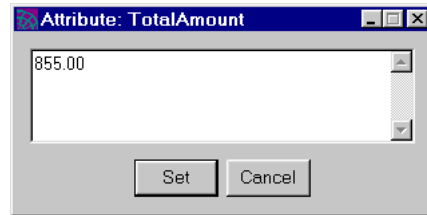
For each process attribute the list displays the attribute name, data type, attribute value, and lock state.

Changing a Process Attribute Value

You might change the value of a process attribute because you want to roll back an activity from ACTIVE to READY state and reoffer the activity using a different attribute-dependent assignment rule.

➤ **To change the value of a process attribute**

1. Select the attribute in any process attribute list window, as described in *“Checking the Value and Lock State of a Process Attribute”* on page 193.
2. Choose Process Attributes > Set. The change value window displays:



3. Enter a new value.
4. Click the Set button.

Removing a Process Attribute Lock

You might remove a process attribute lock because you want to force a change in the value of an attribute, or because a lock was inadvertently not released during an activity rollback or abort.

➤ **To remove a process attribute lock**

1. Select the attribute in any process attribute list window, as described in *“Checking the Value and Lock State of a Process Attribute”* on page 193.
2. Choose Process Attributes > Remove Read Lock

-or-

Process Attributes > Remove Write Lock, depending on the lock state of the process attribute.

Checking for Bottlenecks in Process Execution

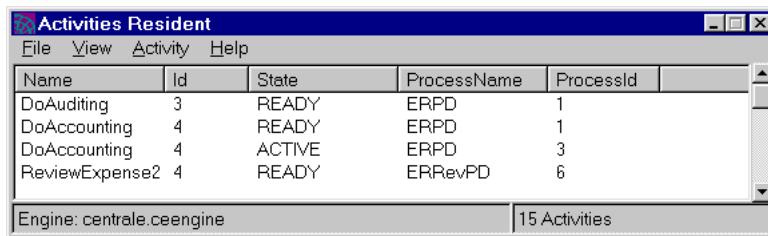
Instances of a given process sometimes stall at a particular activity, causing a bottleneck in execution of that activity. This could happen, for example, if a particular activity were backlogged, or a trigger condition could not be fulfilled.

► To find a process bottleneck

1. Select the engine executing the process in the iIS Console main window.

The engine must be running.

2. Choose Monitor > Activities Resident. This opens the Activity Resident window, displaying the list of activity instances being executed by the engine.



Name	Id	State	ProcessName	ProcessId
DoAuditing	3	READY	ERPD	1
DoAccounting	4	READY	ERPD	1
DoAccounting	4	ACTIVE	ERPD	3
ReviewExpense2	4	READY	ERRevPD	6

Engine: centrale.ceengine | 15 Activities

For each activity instance, the list shows the activity name, activity ID, activity state, process name, and process ID.

3. Filter the activity list by the process name (registered process definition) of the process you are checking (see [“Filtering iIS Console Lists”](#) on page 73).

This filter should list activities for all active process instances corresponding to the registered process.

4. Look for an activity name with more than an average number of instances.

You can also filter the activity list by activity name and check the number of items in the list, which is displayed in the status bar.

5. Filter the activity list by the name of the suspected bottleneck activity.
6. Check the state of activities in the listing.

Depending on the situation, you might need to do further analysis. For example, if most instances of the activity are in READY state, there might currently be too few sessions to do the work or the sessions to which the activity is assigned are doing other work. If most of the activities are in PENDING state, then probably a trigger condition is not being met.

To analyze the situation further you might have to open particular activity instances, check sessions, or possibly check attributes of the corresponding process instances.

Analyzing Process Execution

The iIS product does not provide tools for analyzing information in the engine database's history log tables that could be useful in managing and reporting on process execution. The history log provides information that can be used to answer the following types of questions:

- How long does it take for processes to complete?

You can determine how long it took for a particular process instance (identified by primary attribute) to go from start to finish, or how long it takes, on average, for process instances to complete. You can analyze this question as a function of some particular process attribute (creator, creation date, invoice value, and so forth).

- Where are the bottlenecks in process execution?

There are several approaches to finding bottlenecks. One is to determine how many process instances are currently in a given stage of completion. For example, you could create a graph of how many process instances have reached each activity, and then look at how this graph changes over time to see if a pattern emerges that indicates a bottleneck. You could also look at how this graph changes as a function of some particular process attribute.

Another approach to finding bottlenecks is to determine how long it takes, on the average, for a process instance to complete each activity. Some might take substantially longer than others. You might also look at how long the activity is in each state (PENDING, READY, ACTIVE) to determine a strategy for resolving the bottleneck.

- How well are activity queues functioning?

For a queued activity, you can see how long the queue currently is or how long it is on the average. You can determine the average length of time that activities remain in the queue, or if the prioritizing of activities in the queue leaves some activities at the bottom of the queue for an unacceptable length of time.

- Worker productivity

You can perform productivity analyses. For example, you can tabulate how many activities have been completed in a given time as a function of the session (user). You can also look at how many activities are currently owned (made active) by each user.

- Quality assurance

You can track the work being done on a particular process instance. For example you can determine who completed a previous step in an activity, tracking the process from one user to another.

History Log Schema

The full engine database schema is documented in the [Appendix B, “Engine Database Schema”](#). [“History Log Tables” on page 347](#) of this Appendix lists the schema of all the history log tables. Using this information, you can write your own tools for querying the database and analyzing historical process execution data.

State Values

Many of the history log tables listed in [“History Log Tables” on page 347](#), have a field that represents the state of a given engine object: process instance, activity, session, and timer. The following table shows the values corresponding to the various states of these objects:

Object	State	Value
Process Instance	PREPARE_CREATE	405
	CREATED	410
	ACTIVE	420
	COMPLETED	430
	PREPARE_ABORT	435
	ABORTED	440

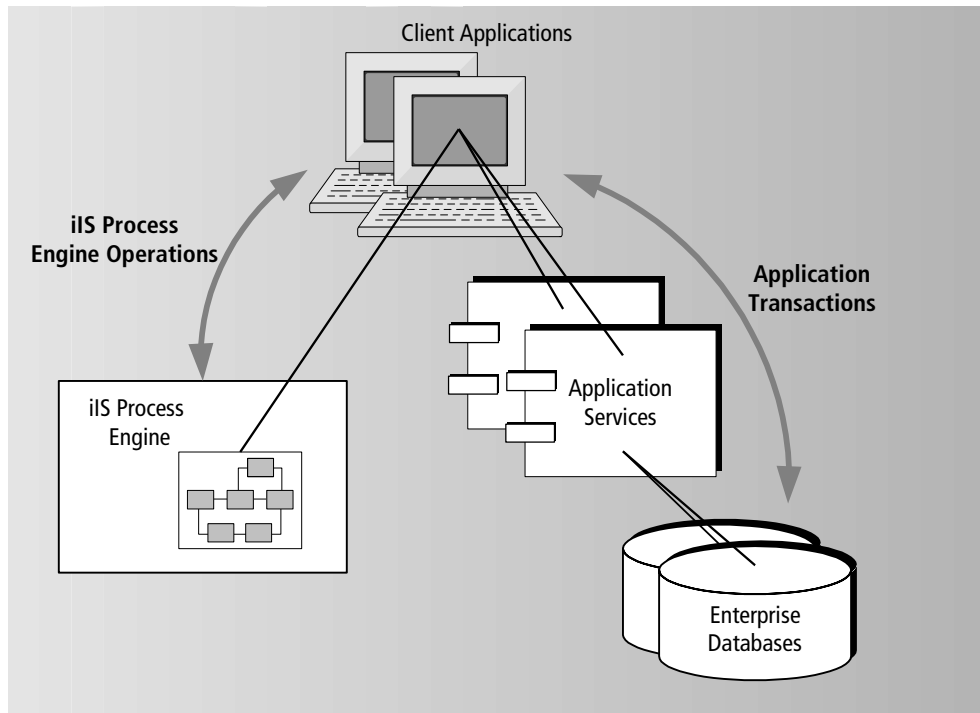
Object	State	Value
Activity	PENDING	10
	READY	20
	PREPARE_ACTIVE	25
	ACTIVE	30
	PREPARE_COMPLETE	35
	COMPLETED	40
	PREPARE_ABORT	45
	ABORTED	50
	DELETED	60
Timer	CREATED	810
	OFF	820
	ON	830
	EXPIRED	840
	DELETED	850
Session	ACTIVE	210
	SUSPENDED	220
	TERMINATED	230

Monitoring and Managing Two-Phase Commit

iIS provides a two-phase commit protocol that allows process client applications to easily synchronize iIS process state changes with application database updates.

NOTE Two-phase commit does not apply to application proxies. This section applies only to client applications that interact directly with the engine.

As illustrated in [Figure 6-3](#), process client applications typically perform iIS operations that change process state maintained by the engine, while also making application database updates.

Figure 6-3 Client Applications Change Both iIS Process State and Application Data

For example, a process client application typically interacts with the engine to start an activity, performs the activity by updating application data, and then interacts with the engine again to complete the activity. If, for some reason, the update of application data fails, the `CompleteActivity` operation should not succeed. Or, similarly, if for some reason the `CompleteActivity` operation fails, the update of application data should not succeed.

The iIS two-phase commit protocol facilitates the synchronization of iIS engine operations with application transactions by letting you place iIS operations in a transactional context. As with application database transactions, the iIS two-phase commit protocol lets you split processing of iIS transactions into two phases: the first phase—*preparing*—guarantees that the operation (transaction) can either commit or roll back; the second phase actually performs the commit (or roll back).

NOTE “Rollback” in this usage is equivalent to “undo.” For example, rolling back a `CompleteActivity` operation cancels the request to complete the activity.

Because of this two-phase commit capability, an iIS engine transaction and an application transaction can be combined into a higher level distributed transaction. The constituent transactions are placed in a PREPARE phase before either can commit. If either of the constituent transactions cannot be placed in a PREPARE phase, then the other is rolled back to its original state. In other words, the distributed transaction only commits if both constituent transactions commit, guaranteeing that application state and process state are synchronized.

The iIS two-phase commit protocol also facilitates synchronization between different activities—or sessions—in a given process instance. For example, if Client Application A (session A) has already prepared an AbortProcess for the process instance, and Client Application B (session B) tries to complete or abort an activity belonging to that process instance, an exception is raised on session B indicating that session A is going to abort the same process instance.

The iIS two-phase commit protocol makes it possible to include the following iIS engine operations in a transactional context:

- creating a process
- aborting a process
- starting an activity
- completing an activity
- aborting an activity

When included in a transactional context, these operations are automatically placed in a PREPARE phase before being explicitly committed or rolled back. For example, any StartActivity or CompleteActivity operation would be automatically placed in a PREPARE phase.

By being placed in a transactional context, these iIS transactions can be synchronized, as described above, with application update transactions. The client application simply prepares any application (database update) operations it needs to make. If the application transaction can be successfully placed in a PREPARE phase, then the client application can commit both the application transaction and the iIS engine transaction. If the application transaction cannot be successfully placed in a PREPARE phase, then the client application should roll back the iIS engine transaction.

All iIS objects associated with a prepared transaction are unavailable for further operations until the client application explicitly commits or rolls back the transaction.

The iIS two-phase commit protocol is implemented by placing an engine session in two-phase commit mode. iIS engine transactions for this session are automatically placed in a PREPARE phase. A session with two-phase commit enabled can only support one iIS transaction at a time.

The iIS process engine uses a unique *transaction ID* to monitor transactions during a two-phase transaction. The client application can specify a unique transaction ID; otherwise the engine generates a transaction ID (based on a time stamp, session name, and the session operation sequence number).

Managing Two-Phase Commit Operations

If a session should be suspended for any of a number of reasons, any iIS transaction in a PREPARE phase is retained on the session, awaiting resolution (commit or rollback). The client application is normally responsible for resolving these transactions; however, in the case of failure, these transactions may be left permanently in the PREPARE phase. Because of this, it is up to a system manager to check for and resolve any transactions left in a PREPARE phase.

For example, a client application may perform an activity, but fail before it can notify the engine to commit the CompleteActivity operation. Similarly, the engine could fail before receiving the commit. Both these situations result in inconsistency of state information between the client application and engine.

To properly resolve iIS transactions in a PREPARE phase, however, you must investigate whether related application transactions were committed or aborted. How you do this depends on the details of the application and how it keeps track of transaction IDs.

You can use Conductor Script commands to identify and resolve transactions in a PREPARE phase, as described in [“Monitoring and Managing Two-Phase Commit Transactions” on page 258](#).

Troubleshooting

This chapter discusses a number of topics and approaches related to diagnosing problems in your iIS process management system.

This chapter covers the following kinds of troubleshooting information:

- alarms
- exceptions
- messages
- performance indicators

Introduction

iIS provides a number of tools to help you diagnose engine problems or problems in process execution with your iIS enterprise applications:

- the Process Engine Alarms window
- the engine component log files
- the iIS Console Trace window
- the Engine Event Trace window
- the iIS Console performance charts

These tools all display various types of information that can be useful in uncovering and diagnosing the cause of problems.

The information displayed in these iIS tools falls into the following categories:

Alarms A number of error conditions in the functioning of an iIS process engine that might require your intervention are reported as alarms. These conditions range from a communication problem between engine components to exceptions generated in the primary engine unit in the course of process execution. Alarms are displayed in the Process Engine Alarm window (see [“Process Engine Alarms Window” on page 205](#)) and written to an alarms table in the engine’s history log database (see [“WFHAlarmLog” on page 349](#)).

Exceptions All error conditions detected by an engine (including alarms) cause an exception to be raised. When the task that encounters the condition originates in the client, the error is reported directly to the client (for example, attempting to start an activity that has already been started). When an error is detected by an internal engine task (for example, attempting to start an unregistered subprocess), the engine records a process exception, activity exception, session exception, or timer exception in the primary engine unit’s log file (see [“Engine Log Files” on page 210](#)). The exception can also be written to the iIS Console trace window (see [“Using the iIS Console Trace Window” on page 215](#)).

Messages Each engine component—whether it be an engine unit, database service, or governor—can generate many types and levels of messages. You can specify the type and level of messages you want to have logged to a component’s log file and to an iIS Console trace window (see [“iIS Console Trace Window” on page 212](#)). You can often use this message output to track down the source of problems.

Performance Indicators You can monitor various aspects of process execution using the iIS Console. For example, you can monitor sessions and executing processes as described in [Chapter 6, “Managing Process Execution.”](#) You can also monitor a number of specific performance indicators using the charting capabilities described in [“Performance Charts” on page 221](#).

This chapter describes how to use the iIS alarms window, log files, tracing window, and performance window to display the above categories of troubleshooting information.

Process Engine Alarms Window

A number of engine conditions result in an alarm. An alarm is a message notifying you of a condition in the functioning of an iIS process engine that might require intervention.

The following list shows these alarms and a description of the condition that gives rise to each:

Alarm	Description
ChannelDisconnect	A communication channel to the engine unit has been disconnected due to the failure of a component or an interruption of the network.
DatabaseDisconnect	A database service has been disconnected due to the partition shutting down or a network failure.
EngineUnitForceStandby	Because an engine unit cannot operate independently in a fault tolerant state, the engine unit has been forced into STANDBY state due to an interruption of contact with its partner.
EngineUnitOnline	The engine unit has gone into ONLINE state and is now capable of accepting client connections.
EngineUnitShutdown	An engine unit has shut down and is no longer running as a partition, due either to operator intervention or to a failure in the software. If a software failure is suspected, look in the log files for the engine units and governor.
Exception	An exception has been generated in the engine unit. The Severity and Message fields contain the information about the exception.
GovernorDisconnect	A communication channel between an engine unit and the Governor has been disconnected due to the failure of a component or an interruption of the network.
ProcessAbort	A process has been aborted. The details field of this alarm gives the ID of the process that was aborted. If this alarm is unexpected, look in the log files for further information.
StartupFailure	An engine unit has failed to start up correctly. Further information as to the reason for the failure can be found in the governor log file. The engine unit is now in STANDBY state.
UserAccess	Access to something has been denied to a user. The Reason field gives the reason for the access violation.

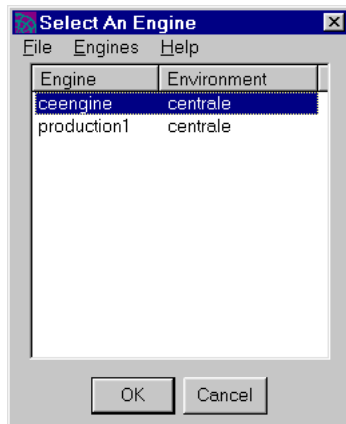
Monitoring Alarms

Alarms for any number of engines can be displayed in the Process Engine Alarms window, allowing you to monitor all your iIS process engines from one central location. (Alarms can also be written to the engine database of any engine for which history logging is turned on.)

To view alarms for a specific engine, you must first register the engine with the alarm service. An engine can be registered with the alarm service before it is started up, and can be unregistered as well.

➤ **To register an engine with the alarm service**

1. Choose Environment > Alarms in the iIS Console main window. The Select an Engine window displays:

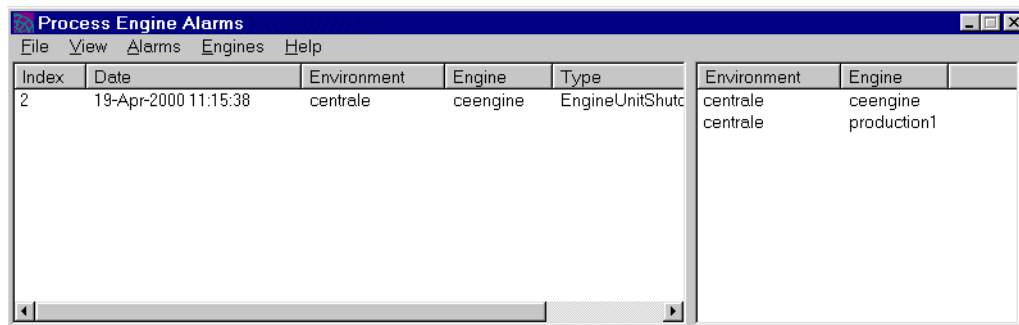


2. Select an engine and click OK.

The engine does not have to be running to register it with the Alarm service.

3. The Process Engine Alarms window displays.

In the following illustration, a registered engine, ceengine, was shut down, causing the EngineUnitShutdown alarm to be displayed.



► To unregister an engine from the Alarm service

1. Select an engine in the engine list in the right hand panel of the Process Engine Alarms window.
2. Choose Engines > Remove Engine.

You can filter the alarms that are displayed in the Process Engine Alarms window, and also search for alarms in the window.

► To filter alarms in the Process Engine Alarms window

1. Choose Alarms > Filter.

The Filter Expression window opens. This window allows you to build an expression to use for filtering the display. Expressions are built using the available attributes listed in the window.

2. In the Filter Expression window, drag an attribute from the Available Attributes list to the Filter Expression editing area.
3. Using a boolean operator together with a legal value for the attribute, build the filter expression, and then click OK.

Attribute values must be legal for the type of the attribute. String values must be enclosed in quotes.

You can add additional expressions to build complex filters. For example, the following expressions displays alarms of type EngineUnitShutdownAlarm for the engine named ceengine:

```
Type="EngineUnitShutdownAlarm"  
Engine="ceengine"
```

For more information on creating filter expressions, refer to [“Filtering iIS Console Lists” on page 73](#).

► **To search for an alarm in the Process Engine Alarms window**

1. Choose Alarms > Find.

The Filter Expression window opens.

2. In the Filter Expression window, build an expression for the alarms you are trying to find.

Use the steps outlined in the previous procedure to build the expression.

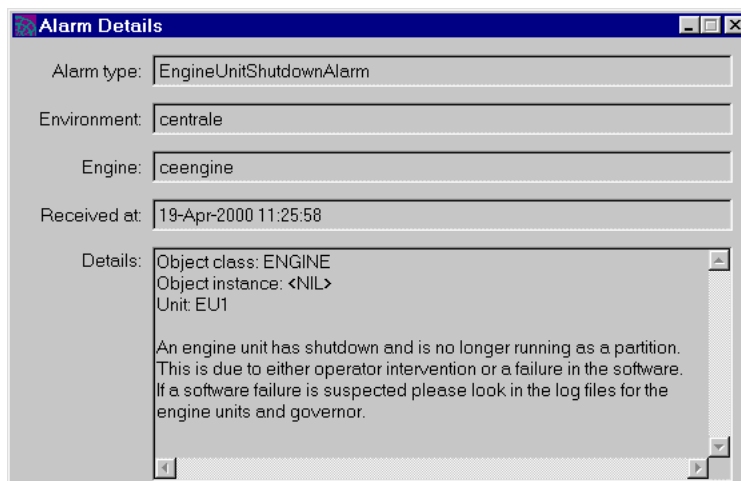
3. Click OK.
4. To search again, using the same expression, choose Alarms > Find Again.

Viewing Alarms

After an engine has been registered with the Alarm service, any alarms sent by that engine (either in start-up or while it is running) are displayed in the Process Engine Alarms window. The Process Engine Alarms window must be open. Alarms that occur when the Process Engine Alarms window is not open, cannot be displayed by opening the window. To see the alarm in that case, you have to open the engine unit log file (see [“Engine Log Files” on page 210](#)).

► **To get detailed information about an alarm**

1. Select the alarm in the Process Engine Alarms window.
2. Right-click and choose Alarm > Open from the popup menu. The Alarm Details window displays:



► **To remove an alarm from the Alarm window**

1. Select the alarm in the Alarms window.
2. Right-click and choose Alarm > Remove from the popup menu.

Engine Log Files

Each iIS process engine component, like other iPlanet UDS partitions, writes to a log file that records exceptions and message output. The component log files are therefore an important source of troubleshooting information.

The log file for each component is written in the FORTE_ROOT/log directory of the node on which the component is executing. Log files are named according to the executing partition's name. Engine components can be executed as standard partitions interpreted by an iPlanet UDS ftexec partition or as compiled partitions. The corresponding log file names and locations on the host server node are as follows:

iPlanet UDS Partition	Standard Output Log File in FORTE_ROOT/log/
Standard partition	forte_ex_process_ID.log (for example, forte_ex_13456.log)
Compiled partition	filename_process_ID.log

Several kinds of information are written to an engine component log file. Some kinds are written automatically and some are under your control. Some information is generated by the underlying iPlanet UDS runtime system, while other information is specific to the execution of iIS processes. In general the information falls into four categories: engine exceptions, requested message output, instrument data, and audit traces.

Engine Exceptions All iIS error conditions detected by an engine cause an appropriate exception to be raised. When the error is detected by an internal engine task (for example, attempting to start an unregistered subprocess), the engine records the exception in the primary engine unit's log file. (It also writes it to the iIS Console trace window, if open.)

Requested Message Output You can request that specific categories of messages be logged to any iIS engine component log file (see [“Messages and Message Filters” on page 212](#)). In general, you can set log filters (or flags) that specify the type and level of messages you want to have logged. (These messages are also written to the iIS Console trace window, if open.)

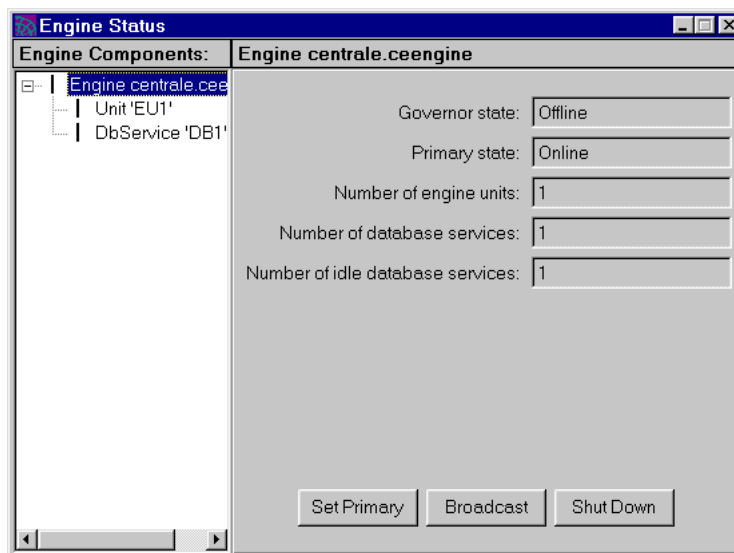
Instrument Data You can decide to have instrument data—such as application resource usage or performance statistics—collected by the underlying system for an engine component and written to the partition log file at time intervals you can specify. The logging of instrument data is determined by values you set for special logging instruments using iPlanet UDS system management tools, such as the iPlanet UDS Environment Console.

Audit Traces Audit traces for important system events (such as starting and stopping partitions, and so forth) are automatically written to log files, depending on the event. You cannot turn off the logging of this information.

Once a component has come on line, you can view its log file from within iIS Console.

► **To view an engine component log file**

1. Select an engine from the list of running engines in the iIS Console main window.
2. Choose Engine > Status. The Engine Status window displays.
3. Select the engine unit for which you want to see the log file.



4. Click the View Partition Log button.

NOTE If an engine component has not come online, its log file cannot be viewed from iIS Console. In that case you need to access the log file directly on the component's host node.

iIS Console Trace Window

iIS Console provides a trace window that displays information written to iIS engine component log files. This information consists of output logged by engine components when they encounter exceptions (error conditions), as well as messages generated in process execution.

Messages and Message Filters

iIS engine components—engine units, database services, and the governor—generate many types and levels of messages. These messages are generated in the course of execution and can be used, in addition to exception output, for tracking down problems.

To view these messages you must specifically request the type and level of message you want to see, and for which engine component. The specified messages are then written to the corresponding component log files (and to the iIS Console Trace window).

You can specify message filters in a number of ways:

- You can set an environment variable (FORTE_LOGGER_SETUP) on each node that specifies the message filters for every iPlanet UDS partition (engine component) started on that node.
- You can set message filters for an individual iPlanet UDS partition (overriding the environment variable—or its default value) by specifying a log flag (-fl) on the partition start-up command. (There is currently no way to do this in iIS Console or Conductor Script.)
- You can set message filters dynamically for a running partition (overriding the specification in a logger flag or environment variable) by using the File > Modify Log Flags command in the trace window of the iIS Console (or the Conductor Script ModLogFlags command, [page 292](#)).

This section discusses how to use the iIS Console Trace window to select and display messages generated by iIS engine components.

Specifying Message Filters

The syntax for specifying log message filters is shown in [Figure 7-1](#). Generally you specify a standard output log file, “%stdout,” as the log file name (see “[Engine Log Files](#)” on page 210). You can specify more than one log file if you want different information written to different files. For each file you can specify any number of filters, each separated by a space.

Figure 7-1 Specifying iPlanet UDS Message Output Filters

FORTE_LOGGER_SETUP (or -fl flag):

“file_name (message_type [:service_type[:group_number[:level_number]]])...”

Message Types	
err	Error messages
sec	Security messages
aud	Audit messages
prf	Performance information
cfg	Configuration modification
trc	Debugging information
*	all of the above

In general, for each log file you can specify four levels of filter, as follows:

Message type (mandatory) You can include more than one message type separated by spaces—the message types available are shown above.

Service type Each message type can be divided into a maximum of ten service types, which typically map to important application services.

Group number Each service type can be divided in turn into a maximum of 63 group numbers, which typically map to a group of facilities.

Level number Each group number can be subdivided into up to 255 additional levels.

iIS Message Filters

iIS message filters follow the general guidelines above and have the following particularities:

Message type trc (debug)

Service type cr (iIS runtime)

Group number iIS uses the following group designations:

Group	Description
1	Engine
2	Session
3	Process
4	Activity
5	Timer
6	currently not used
7	History
8	Access checking
9	Recovery
10	Registration
13	Client application messages

Level number iIS uses the following level designations:

Group	Description
2	Coarse
3	Medium
4	Fine
255	Method entry and exit (where implemented)

For example, an iIS message filter for a primary engine unit might be as follows:

```
FORTE_LOGGER_SETUP: "%stdout (trc:cr:2:2) "
```

This filter specifies that all messages about sessions created in the engine—but not a lot of detail—be written to the primary engine unit’s log file (on the server node on which the primary engine unit is running) and to the iIS Console trace window (if you have opened it at least once before).

If you do not specify any iIS message filters, none are set by default.

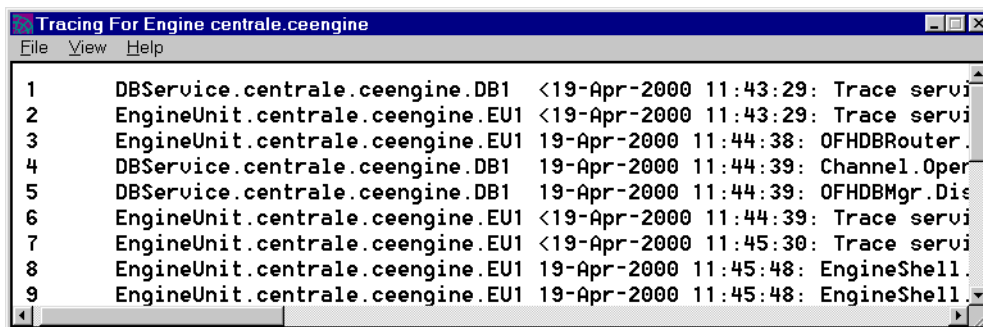
Using the iIS Console Trace Window

You can use the iIS Console trace window to select and display messages generated by iIS process engine components and to view exception output. Using the trace window, you can monitor engines distributed throughout your environment from one central location.

► To open the iIS Console Trace window

1. Select a running engine in the iIS Console main window.
2. Choose Engine > Trace. This opens the iIS Console Trace window

The trace window shows all messages you specify for components of the selected engine.

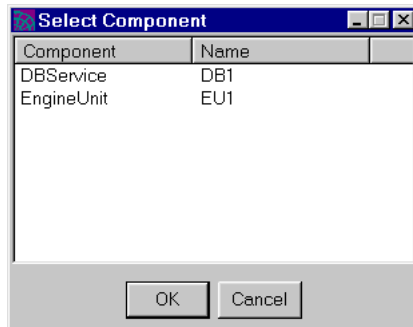


Setting Message Filters

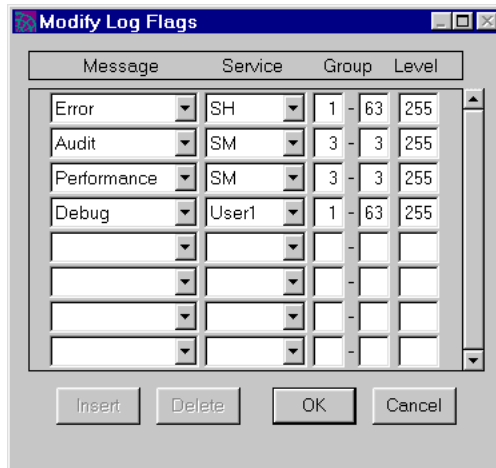
The iIS Console provides a Modify Log Flags window for setting the message filters described in the previous section, *"iIS Message Filters."*

► **To set message filters for a selected engine**

1. Select a running engine in the iIS Console main window.
2. Choose Engine > Trace. The iIS Console Trace window displays.
3. Choose File > Modify Log Flags. This opens the Select Component window.



4. Select an engine component and click OK to open the Modify Log Flags window.



5. Add, modify, or delete a message filter and click OK.

Use the message filter specifications documented in the previous section, [“iIS Message Filters.”](#)

NOTE To specify the **trc** message type, select Debug in the Message column of the Modify Log Flags dialog.

All messages generated by the component that meet the specification are now written to the trace window, as well as to the components log file.

Special Example: Write Client Messages to Trace Window

As an example of using message filters, suppose you want to have all messages generated by client applications written to the iIS Console trace window. To do this, the client applications must generate the messages and pass them to the engine to be written to the engine log file and trace window. To pass the messages to the engine, the client partitions must also set their trace flag to the value shown in the procedure below. (This is a special procedure used for troubleshooting and diagnosing client application problems.)

► **To set the engine tracing**

1. Select the engine in the iIS Console main window.
2. Choose Engine > Trace. The iIS Console Trace window displays.
3. Choose File > Modify Log Flags.
4. In the Modify Log Flags window, enter the following filter: Debug:CR:13:4
 This filter causes all messages generated by client applications that have this same trace flag set to be written to the engine trace window and the primary engine unit log file.
5. For each client partition whose messages you want to trace in iIS Console, set the following filter: Debug:CR:13:4

iIS Console Engine Event Filter Window

iIS Console provides a filtering mechanism and trace window to let you view information about iIS process engine events. These messages are generated in the course of process execution and can be used, in addition to exception output, for tracking down problems.

When you trace engine events for a specified engine you can:

- view all engine events for that engine
- view all engine events belonging to an existing process instance
- view all engine events belonging to the next process instance of a specified process definition
- view engine events according to a custom filter you specify

Engine Event Types

iIS Console engine event types consist of:

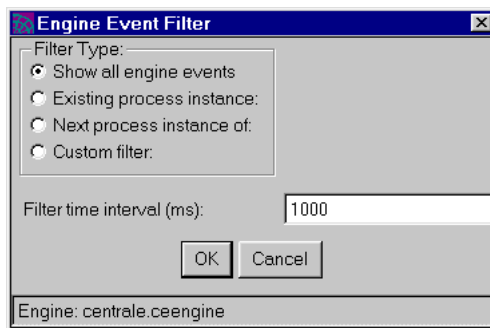
- timer state changes (on/off/value change)
- session events (start, suspend, terminate)
- activity state changes
- process events (start, terminate, new registration, deregistration)
- security events (access denial for invalid logon)
- engine state changes (start, standby, shutdown)

Using the iIS Console Engine Event Filter Window

You can use the iIS Console engine event filter window to specify and display messages regarding engine event generated by any iIS process engine in your environment.

➤ **To open the Engine Event Filter window**

1. Select a running engine in the iIS Console main window.
2. Choose Engine > Event Trace. This opens the Engine Event Filter window.



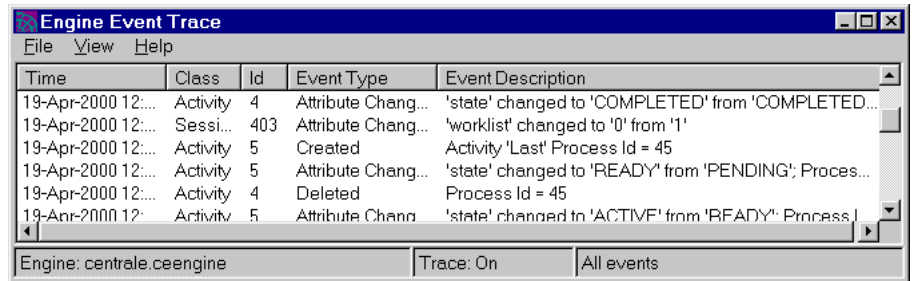
Displaying All Engine Events

You can view all engine events for a running engine.

► To view all engine events for the selected engine

1. From the Engine Event Filter window, select the option Show all engine events.

Each engine event is identified by event type, class, ID, description, and time of occurrence. The information starts displaying after the next engine event occurs.

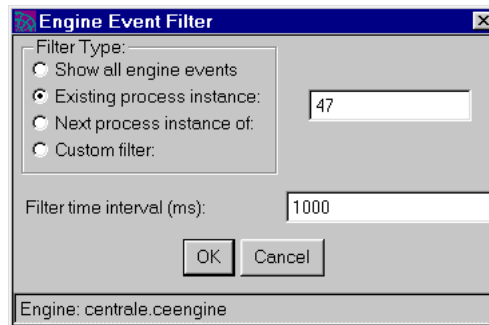


Displaying Process Instance Events

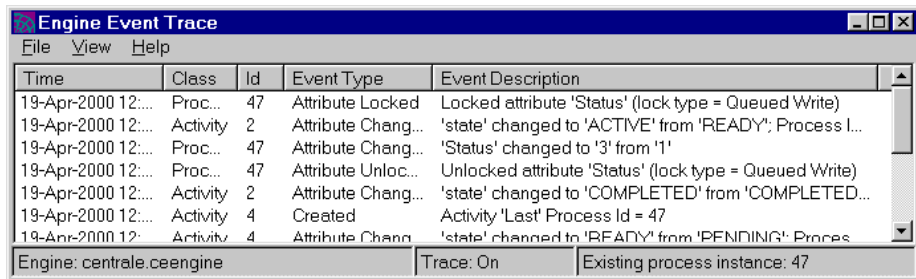
You can display all engine events for a given process instance, either one that exists, or the next process instance for a given process definition in your repository.

► To view all engine events for an existing process

1. From the Engine Event Filter window, select the option Existing process instance: and enter the ID for that process instance.

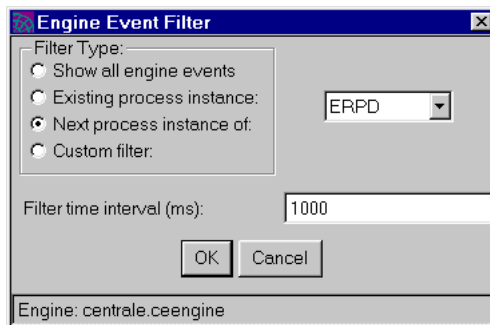


Events for the specified process instance display:

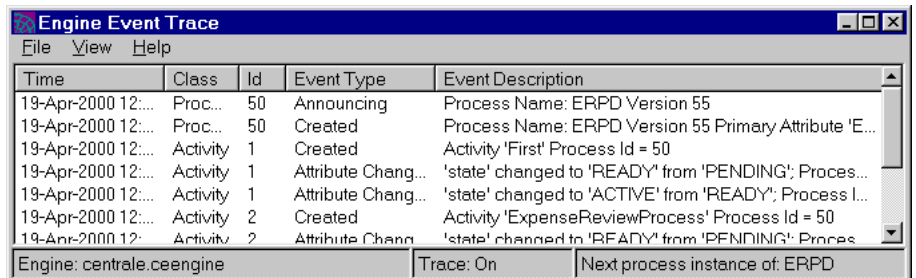


➤ **To view all engine events for the next process instance of a specific process definition**

1. From the Engine Event Filter window, select the option Next process instance of: and select a process definition from the drop down list.



All engine events for the next process instance of the process definition will start displaying when the process instance starts. Each engine event is identified by event type, class, ID, description, and time of occurrence.

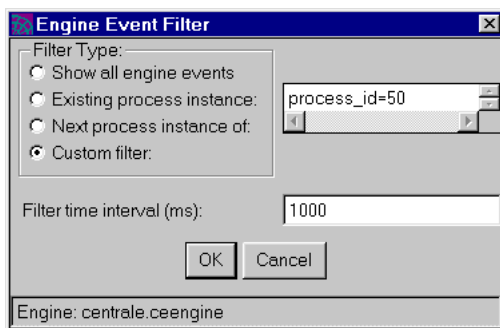


Filtering Engine Events

A fourth way to view engine events is by providing a custom filter. You can filter on events involving sessions, process instances, activities, engine objects, timers and registrations. For details on valid filter expressions, see [“CreateFilter” on page 279](#).

► To specify a custom filter

1. Select the option Custom filter and enter a valid filter expression.



Events which satisfy the filter expression are displayed:

Time	Class	Id	Event Type	Event Description
19-Apr-2000 12:...	Activity	3	Created	Activity 'DoAuditing' Process Id = 50
19-Apr-2000 12:...	Activity	3	Attribute Chang...	'state' changed to 'READY' from 'PENDING'; Proces...
19-Apr-2000 12:...	Activity	4	Created	Activity 'DoAccounting' Process Id = 50
19-Apr-2000 12:...	Activity	4	Attribute Chang...	'state' changed to 'READY' from 'PENDING'; Proces...
19-Apr-2000 12:...	Activity	2	Deleted	Process Id = 50
19-Apr-2000 12:...	Activity	2	Attribute Chang...	'state' changed to 'COMPLETED' from 'DELETED'; P...

Engine: centrale.ceengine Trace: On Custom filter: process_id=50

Performance Charts

An iIS process engine that is performing process execution without any visible sign of problems might nevertheless encounter performance bottlenecks. One cause might be increased load on the engine.

It is therefore a good idea to regularly monitor a number of performance indicators—instrumentation within an engine that can be displayed over time in the iIS Console Performance Charts. The instruments displayed in the Performance Charts are the following:

Instrument	Description
Process Instances	number of active process instances
Activities	number of activities in the ready or active state
Average response time	the average time it takes for the engine to complete a request made by a session (such as OpenSession, StartActivity, and CompleteActivity)
Average transaction commit time	the time it takes the engine to commit a change of state (complete a transaction involving the engine database)
Allocated memory	the amount of the active memory currently allocated to objects created by the primary engine unit
Peak allocated memory	the amount of allocated memory remaining after the most recent memory reclamation (probably the best measure of active memory utilization)

These instruments indicate how performance might be impacted by an increasing load on the engine.

For example, as the number of sessions increases over time, you would expect the average response time to increase. At some point, you might decide to add an additional engine to your workflow system to help balance the load.

As the number of process instances and activities increase over time, you would expect the average transaction commit time to increase. In this case you might decide to start up additional database services to speed throughput to the engine database.

In many situations these factors have complicated interactions and you have to monitor these indicators carefully under varying conditions, analyzing the source of any performance bottlenecks.

Viewing Performance Indicators

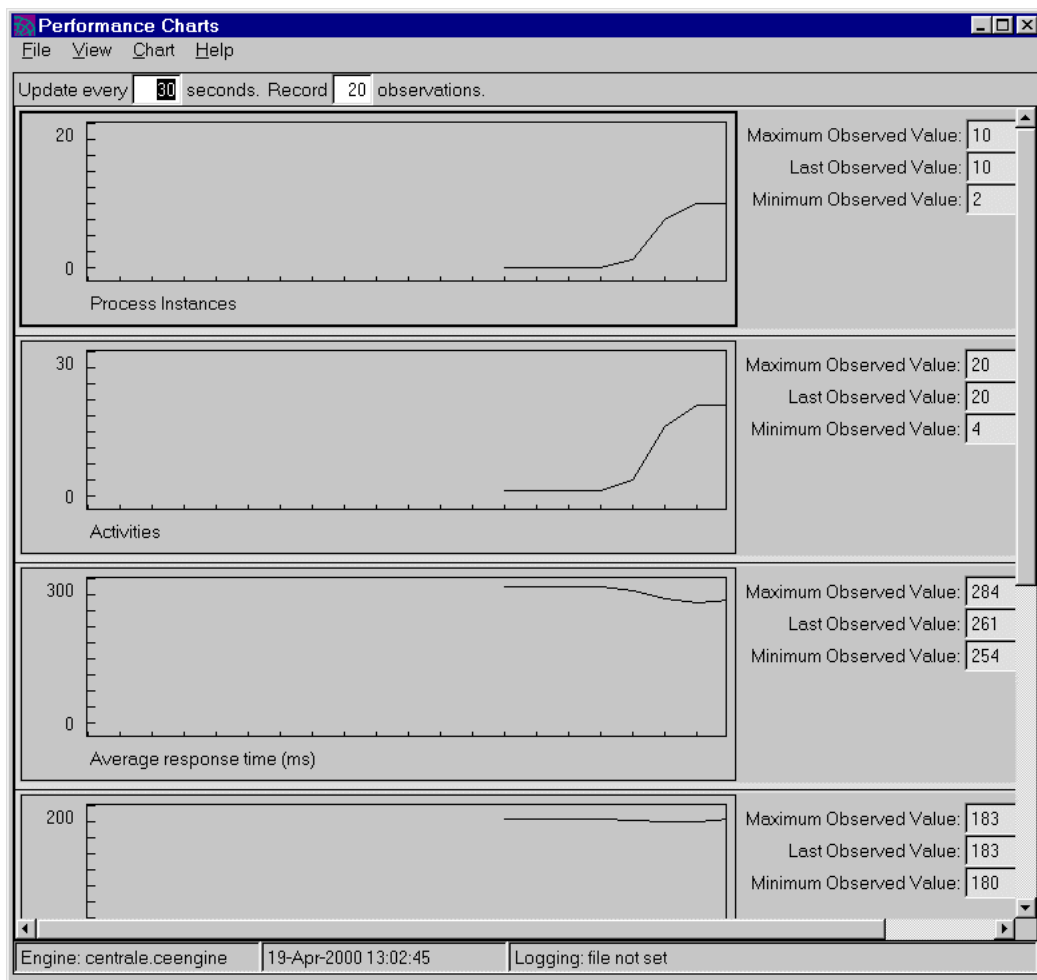
You can view performance indicators for any running engine in the Performance window.

► **To view performance indicators for an engine**

1. Select a running engine in the iIS Console main window.
2. Choose Monitor > Performance.

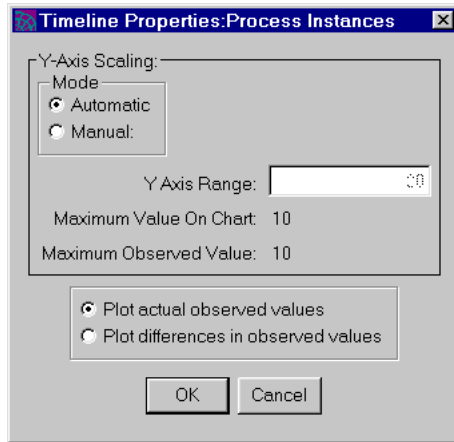
The Performance Charts window is displayed. You can choose View > Observed Values to expand the Charts window to display maximum, last observed, and minimum values for the charts.

3. You can change the interval between each update to the chart by changing the Update Every ___ seconds field. You can also change the number of entries recorded on the chart by changing the Record ___ Observations field.



4. To set the vertical scale parameters for the chart and see the maximum and minimum values recorded by this chart, click somewhere in the chart, then right-click and choose Chart > Properties from the popup menu.

The Timeline Properties window displays. You can choose to display observed values or the differences between succeeding values.



5. To modify the appearance of the chart, click somewhere in the chart, then right-click and choose one of the following commands from the popup menu:
 - o Chart > Line Color
 - o Chart > Line Weight
 - o Chart > Set Default

Logging Performance Information

You can save performance information to a log file.

► **To log performance information**

1. At the Performance Charts window, choose File > Set Logging File.
2. Define a name and location for the performance log data file.
3. Click OK.
4. Choose File > Start Logging to File.

► **To stop logging performance information**

At the Performance Charts window, choose File > Stop Logging to File.

Using the Conductor Script Utility

This chapter describes the Conductor Script utility, the iIS command line system management interface.

The Conductor Script utility is the command line equivalent of the iIS Console. However, Conductor Script also lets you incorporate system management tasks into scripts. These scripts can then be executed at engine startup, regular intervals, or other times to register iIS distributions, or to start and shut down engine components.

This chapter covers the following topics:

- starting and exiting Conductor Script
- working with Conductor Script
- using Conductor Script to perform system management tasks

For a complete reference listing of all Conductor Script commands, see [Appendix A, “Conductor Script Commands.”](#)

Overview

With the exception of configuring iIS process engines, the Conductor Script utility is the functional equivalent of the iIS Console: you can use it to perform iIS process management tasks, such as managing iIS engines, managing registration, and managing process execution.

Like the iIS Console, Conductor Script connects to and communicates with the executing Environment Manager and any active Node Managers in your environment. It lets you perform iIS process management tasks by providing you with access to your iIS engines and to the full hierarchy of iPlanet UDS system management agents.

Unlike the iIS Console, however, you can use Conductor Script to perform system management by building and executing predefined scripts of Conductor Script commands. For information on building and executing Conductor Script scripts, see [“Writing and Executing Scripts” on page 232](#).

This chapter explains how to perform system management tasks using Conductor Script. It provides a list of Conductor Script commands for various sets of tasks, and in some cases provides detailed step by step procedures. However, it does not provide background and conceptual information offered elsewhere in this manual.

For a complete reference listing of all Conductor Script commands, refer to [Appendix A, “Conductor Script Commands.”](#)

Conductor Script Help

Conductor Script provides an online help utility, which provides help for all Conductor Script commands. At a Cscript prompt type Help to list all the available commands. For help on a specific command, type Help plus the command. You can also use wildcard characters to get help on groups of commands.

NOTE Some commands are associated with a particular engine component, so help for those commands is only available when an engine or the particular component is current.

For information on the Help command, see [“General Conductor Script Operations” on page 231](#).

Starting Conductor Script

You can start Conductor Script on any node in your iPlanet UDS environment where it has been installed.

- **To start Conductor Script on Windows or Windows NT**
 1. Double-click the Conductor Script icon.
 2. Enter a valid password if one is requested (see [“SetPassword” on page 300](#)).

- **To start Conductor Script on UNIX, OpenVMS, or Windows NT**
 1. Use the `cscript` command (see “Using the Cscript Command” below for information).
 2. Enter a valid password if one is requested (see “SetPassword” on page 300).

When Conductor Script starts, it opens a `cscript` command prompt.

Using the Cscript Command

As mentioned above, you can start iIS Console on command line-based operating systems by executing the `cscript` command.

The syntax of the `cscript` command for most platforms:

```
cscript [-fl message_filters] [-fm memory_options] [-i input_file]
[-o output_file] [-fns name_service_address]
```

The syntax of the `cscript` command for OpenVMS:

```
VFORTE CSCRIPT
  [/LOGGER=message_filters]
  [/MEMORY=memory_options]
  [/INPUT=input_file]
  [/OUTPUT=output_file]
  [/NAMESERVER=name_service_address]
```

As in all iPlanet UDS command line specifications, if you use a name that includes a space, you should enclose the name in double quotation marks.

The following table describes the command line flags for the `cscript` command.

Flag	Description
<code>-fl message_filters</code> <code>/LOGGER=message_filters</code>	Specifies the log flags to use for the Conductor Script session. See the <i>iPlanet UDS System Management Guide</i> for information about the syntax for specifying log flags. Overrides the <code>FORTE_LOGGER_SETUP</code> environment variable setting. On UNIX, you must specify the log flags in double quotes.

Flag	Description
-fm <i>memory_options</i> /MEMORY= <i>memory_options</i>	Specifies the memory flags to use for the Conductor Script session. See the <i>iPlanet UDS System Management Guide</i> for syntax information. Overrides defaults appropriate for the operating system. On UNIX, you must specify the memory flags in double quotes.
-i <i>input_file</i> /INPUT= <i>input_file</i>	Specifies an input file. The file should consist of a Conductor Script script—a set of Conductor Script commands—that you want to execute automatically when the Conductor Script utility starts.
-o <i>output_file</i> /OUTPUT= <i>output_file</i>	Specifies an alternate output file, in addition to stdout.
-fns <i>name_service_address</i> /NAMESERVER= <i>name_service_address</i>	Specifies the name service address for the environment in which this application will run. This value overrides the value, if any, specified by the FORTE_NS_ADDRESS environment variable. If you want your application to be able to switch to a backup Environment Manager if the primary Environment Manager fails, you can also specify multiple name service addresses, as discussed in the <i>iPlanet UDS System Management Guide</i> .

Working with Conductor Script

Conductor Script is a command line utility built on top of the iPlanet UDS Fscript command utility. As such, it includes Fscript commands for performing repository-based operations, as well as a large number of iIS-specific commands. (A few Fscript commands, however, have been disabled in Conductor Script.)

This section covers topics related to some of the general purpose Fscript commands accessible from within Conductor Script. iIS-specific operations and commands are discussed in subsequent sections. The topics covered in this section are the following:

- general Conductor Script operations
- writing and executing scripts
- operating system and file management commands

For full reference documentation of the commands discussed below, consult the *Fscript Reference Guide*.

General Conductor Script Operations

The following table lists Conductor Script commands used for general operations such as getting online help, defining the format of file names, specifying a directory search path, and so on:

Command	Arguments	Function
AddAlias	<i>alias_name</i> <i>command_string</i>	Define an alias for a Conductor Script command and its arguments.
AddPath	<i>directory_name</i> [; <i>directory_name</i> ...]	Add the specified directories to the current search path (see SetPath).
Exit	—	Exit Conductor Script.
Help	[<i>command_name</i> <i>match_string</i>]	List help for general Conductor Script commands and currently available FNscript commands. Use a wildcard (*) to get help on similarly named commands.
ModLogger	+ (<i>log_flags</i>) / - (<i>log_flags</i>)	Modify the current iPlanet UDS log message filters (log flag settings) for Conductor Script.
Quit	—	Exit Conductor Script.
RemoveAlias	[<i>alias_name</i>]	Remove an alias definition.
SetOutFile	[<i>file_name</i>]	Specify a file where standard output is printed.
SetPath	<i>directory_name</i> [; <i>directory_name</i> ...]	Set the directory search path used by any of the commands that take a file name as an argument.
ShowAlias	[<i>alias_name</i>]	Display one or all defined aliases with their expansions.
ShowPath	—	Show the current directory search path (see SetPath).

Command	Arguments	Function
UseLocal	—	Set Conductor Script to recognize file names specified in local operating system format.
UsePortable	—	Set Conductor Script to recognize file names specified in iPlanet UDS portable name format.

Writing and Executing Scripts

This section describes Conductor Script commands you use to write and execute scripts to automate iIS process management tasks. For full reference documentation of the commands, consult the *Fscript Reference Guide*.

One of the advantages of using Conductor Script over the iIS Console is that Conductor Script lets you automate your routine system management tasks. You can capture a sequence of Conductor Script operations into a script file and then run that file at a later time.

You can run a script either by starting Conductor Script with the **-i** flag (and supplying the script file name) or by starting Conductor Script and then issuing the **Include** command when you want to run the script.

Comments

To include comments in your scripts, start the line containing the comment with the **#** character, as shown in the following example:

```
# Shut down the Banking engine.
FindEngine Banking
Shutdown
```


The following table lists Conductor Script commands most often used for writing and executing scripts:

Command	Arguments	Function
CommentOff	—	Stop writing script file commands and output to standard output.
CommentOn	—	Write script file commands and output to standard output.
Delay	<i>milliseconds</i>	Delay execution of the next command for the specified number of milliseconds.
Include	<i>filename</i>	Execute the commands in a specified script file.
Repeat	<i>repeat_count</i>	Repeat execution of the next command the specified number of times.
Script	<i>filename</i>	Capture Conductor Script commands and write them into a specified script file.
ShowExpansions	<i>show_flag</i>	Enable or disable the printing of alias expansions to standard output when they occur.
SilentOff	—	Turn off printing of exceptions to standard output.
SilentOn	—	Turn on printing of exceptions to standard output.
Step	—	Step through the commands in an include script, prompting you for each command.

Operating System and File Management Commands

The following table lists Conductor Script commands for interacting with the operating system on a node to perform file management operations.

Command	Arguments	Function
Cd	<i>directory_name</i>	Change the current working directory.
Chmod	<i>mode</i> <i>file_name</i> [<i>r</i>]	Change the access permissions of the specified file.

Command	Arguments	Function
CopyFile	<i>file1_name</i> <i>file2_name</i> [r]	Copy a specified file in the local file system.
Cp	<i>file1_name</i> <i>file2_name</i> [r]	Copy a specified file in the local file system.
Directory	<i>directory_name</i>	List the files in a directory.
Duplicate	<i>file1_name</i> <i>file2_name</i> [r]	Copy a specified file in the local file system.
ExecCmd	<i>opsys_cmd</i> [bg_flag] [in_file] [out_file] [err_file]	Execute the specified operating system command.
ExecLocal	<i>opsys_cmd</i>	Execute the specified operating system command (for path names in <i>opsys_cmd</i> that use local format).
ExecPortable	<i>opsys_cmd</i>	Execute the specified operating system command (for path names in <i>opsys_cmd</i> that use portable format)
ListFile	<i>ListFile</i>	Lists the contents of the specified file onto standard output.
ListFiles	<i>directory_name</i>	List the files in a directory.
Ls	<i>directory_name</i>	List the files in a directory.
Mkdir	<i>directory_name</i>	Make a new directory.
Mv	<i>old_file_name</i> <i>new_file_name</i>	Rename a specified file in the local file system.
Pwd	—	Display the name of the current working directory.
ReadIntoFile	<i>file_name</i> [term_str]	Read subsequent lines and write them to the specified file until the terminating string is encountered.
Rm	<i>file_name</i>	Remove a specified file in the local file system.
SetDefault	<i>directory_name</i>	Change the current working directory.
WhichFile	<i>file_name</i>	Find the specified file in the current directory search path.

Managing iIS Process Engines with Conductor Script

This section describes how to perform the following engine management functions using Conductor Script:

- starting an engine
- monitoring an engine
- changing engine component states (including stopping an engine)

Conductor Script cannot be used to configure an engine. You can use iIS Console (see [“Configuring an Engine” on page 85](#)) or edit an engine configuration file by hand (using the information in [“Engine Configuration File” on page 90](#)).

For background information on starting an engine, monitoring it, and changing component states, as well as how to perform these functions using iIS Console, see [Chapter 4, “Managing Engines.”](#)

Starting an Engine

You can start engine components from a centralized location using the Conductor Script utility (see [Appendix A, “Conductor Script Commands”](#)). Conductor Script uses iPlanet UDS system management agents to start the partitions corresponding to each of the engine components, as specified in the engine configuration file.

► To start an engine

1. Enter `ListEngines` to get a list of configured engines.

The list indicates which engines are online.

2. Use the `FindEngine` command to choose the engine you want to start.

The selected engine is the current engine.

3. Use the `StartEngine` command to start the current engine.

The engine starts using the component partitioning and startup information in the engine configuration file (see [“Engine Configuration File” on page 90](#)). The components are started in the following order: governor, engine units, database services.

Starting Individual Engine Components

You can start an individual engine component on any node on which it is installed. Each component is an application consisting of a single server partition that must establish communication channels with other components.

Except for the governor, as each of these partitions is started, it must be provided a component name and the name of the engine to which it belongs. The partition uses this engine name along with its component name to explicitly register itself with the iPlanet UDS Name Service. As each component starts up, it looks in the Name Service registry for the names of other components with which it must establish a communication channel. When all necessary communication channels are established, the primary engine unit can log on to its database and perform other operations needed to come fully online.

The engine startup protocol is designed to allow components to start in any order. They simply wait for the other components to which they must connect. However, because an engine unit will eventually time out if it cannot log on to its database or establish a communication channel with its partner or the governor, there is a preferred order for engine startup if there might be a delay in starting a component.

- **To start an engine where delays might be involved**
 1. Start the governor.
 2. Start at least one database service.
 3. Start each engine unit.

- **To start an individual engine component (governor, database service, or engine unit)**
 1. Use the `FindEngine` command to the engine you want to start to the current engine.
 2. Use the `FindNode` command to set node for the component to the current node.
 3. Use the appropriate Conductor Script `start` command to start the engine component on the current node.

The Conductor Script commands used to start engines are documented in [Appendix A, “Conductor Script Commands”](#) and summarized in the following table:

Command	Arguments	Function	See...
ListEngines	—	Display a list of all running and configured iIS process engines in the environment.	page 289
FindEngine	<i>engine_name</i>	Set specified engine as the “current” engine. The engine (nor any of its components) does not have to be running.	page 284
StartEngine	[newLog] [newState] [newRegistration] [cold]	Start all components of the engine as specified in the engine configuration file. (Includes various start options that create new engine database tables.)	page 315
FindNode	<i>node_name</i>	Set the specified node as the “current” node. The node must exist in the environment.	page 285
StartGovernor	—	Start the governor for the current engine on the current node.	page 316
StartDBService	<i>service_name</i> <i>priority</i>	Start the specified database service for the current engine on the current node.	page 315
StartUnit	<i>unit_name</i> [cold]	Start the specified engine unit for the current engine on the current node. (Includes cold start option to create engine database.)	page 317

As indicated in the table, the engine unit partition has a startup argument which specifies that when the engine unit comes ONLINE, that it create the required tables in the engine database. This argument should be used when you start an engine for the first time, or if you explicitly want to delete the existing database schema.

NOTE A cold start purges and deletes any tables that already exist, including the registration table and the current state and history log tables. Be careful when using this option, since you can lose unrecoverable information.

Example Manual Startup Scenario

You can override the component partitioning scheme in the engine configuration file by starting each engine component manually. As an example, take the engine partitioning scheme illustrated in [Figure 4-2 on page 88](#). Assume the iPlanet UDS environment is named “Galaxy”, and the engine is named “Enterprise.” The partitioning scheme is summarized in the following table:

Component	Component Name	Location (Node Name)
Governor		Server1
Primary engine unit	primero	Server2
Backup engine unit	secundo	Server3
Database service	main1 (priority=10)	Server4
Database service	main2 (priority=10)	Server4
Database service	main3 (priority=10)	Server4
Database service	load1 (priority=9)	Server1
Database service	load2 (priority=8)	Server3

The following Conductor Script commands start up the Enterprise engine in the preferred order.

```
findengine galaxy.enterprise
findnode server1
startgovernor
startdbservice load1 9
findnode server2
startunit primero
findnode server3
startunit secundo
startdbservice load2 8
findnode server4
```

Monitoring Engines and Engine Components

After you have started an engine, you should ensure that it has started successfully and monitor it periodically.

You can get status information about the engine as a whole and about each of the individual engine components. Conductor Script uses iIS engine agents to gather engine component data. Each agent has a set of instruments that report relevant information.

Monitoring the Engine

Typically you want to know which engine components are up and running.

➤ **To check the engine runtime configuration**

1. Use the `ListEngines` command to see which engines are running in your environment.
2. Use the `FindEngine` command to make the engine of interest current.
3. Use the `ShowStatus` command to display the runtime configuration of the current engine.

The Conductor Script commands used to monitor engines are documented in [Appendix A, “Conductor Script Commands”](#) and summarized in the following table:

Command	Arguments	Function	See...
FindEngine	<i>engine_name</i>	Set specified engine as the “current” engine. The engine (nor any of its components) does not have to be running.	page 284
ListEngines	—	Display a list of all running and configured iIS process engines in the environment.	page 289
ShowEngine	<i>engine_name</i>	Display status of the specified engine.	page 306
ShowStatus	[short]	Display status of the current engine. (No engine components need be running for this command to work.)	page 309

Monitoring Individual Engine Components

You can get more information about individual engine components.

► To monitor individual engine components

1. Use the `ListEngines` command to see which engines are running in your environment.
2. Use the `FindEngine` command to make a particular engine current.
3. Use the appropriate `FindComponent` command to make a component current.
4. Use the `ShowStatus` command to display information about the current component.

The Conductor Script commands used to monitor individual engine components are documented in [Appendix A, “Conductor Script Commands”](#) and summarized in the following table:

Command	Arguments	Function	See...
FindEngine	<i>engine_name</i>	Set specified engine as the “current” engine. The engine (nor any of its components) does not have to be running.	page 284
FindDBService	<i>service_name</i>	Set the specified database service for the current engine as the current component. The database service must be running.	page 283
FindGovernor	—	Set the governor for the current engine as the current component. The governor must be running.	page 284
FindPrimary	—	Set the primary engine unit for the current engine as the current component. The primary unit must be ONLINE.	page 285
FindUnit	<i>unit_name</i>	Set the specified engine unit for the current engine as the current component. The engine unit must be running.	page 286
ListEngines	—	Display a list of all running and configured iIS process engines in the environment.	page 289
ShowStatus	[short]	Display the status of the current component. If the current component is an engine unit in the ONLINE state, then this command shows the full instrumentation implemented for all the engine’s internal manager objects.	page 309

Changing Engine States

You can change the state of an engine by changing the state of an engine unit or by shutting down an engine component.

Changing Engine Unit States

You might want to change the state of an engine unit for a number of reasons. For example, you might want to close all client sessions with a primary engine unit (change its state from ONLINE to STANDBY) in order to perform administrative functions on the database. Or you might want to place a primary engine unit in STANDBY state so you can make its partner primary. When the partner goes from STANDBY to ONLINE, it recovers state information from the database.

► To change the state of an engine unit

1. Use the `FindEngine` command to make the engine of interest current.
2. Use the `FindUnit` or `FindPrimary` command to make the component of interest current.
3. Use the `SetState` command to set the engine unit state.

If you want to place the primary unit on standby, and the backup unit online, you can perform this operation in a single command using the `SetPrimary` command.

The Conductor Script commands used to change engine states are documented in [Appendix A, “Conductor Script Commands”](#) and summarized in the following table:

Command	Arguments	Function	See...
<code>FindEngine</code>	<i>engine_name</i>	Set specified engine as the “current” engine. The engine (nor any of its components) does not have to be running.	page 284
<code>FindUnit</code>	<i>unit_name</i>	Set the specified engine unit for the current engine as the current component. The engine unit must be running.	page 286
<code>FindPrimary</code>	—	Set the primary engine unit for the current engine as the current component. The primary unit must be ONLINE.	page 285
<code>SetState</code>	<i>state</i>	Set the state for the current engine unit to ONLINE or STANDBY.	page 302

Command	Arguments	Function	See...
SetPrimary	<i>unit_name</i>	Set the specified engine unit for the current engine as the primary unit. This will change the states of the engine units to make the specified unit ONLINE.	page 301

Shutting Down Engine Components

You can shut down an individual component or all engine components using the `Shutdown` command. This command shuts down the current component or engine.

If your engine components are standard iPlanet UDS partitions (interpreted by an iPlanet UDS `ftexec`) shutting down a component leaves the corresponding `ftexec` running.

Managing Registrations with Conductor Script

This section describes how to perform the following registration functions using Conductor Script:

- making iIS library distributions
- registering iIS distributions
- unregistering iIS distributions

For background information on iIS library distributions and registration, including how to perform registration using iIS Console, see [Chapter 5, “Managing Registrations.”](#)

Making iIS Library Distributions

You can use Conductor Script to perform operations normally performed through the iIS process development workshops, the operations performed with the `File > Compile` and `File > Distribute` commands in the Process Development, Assignment Rule, User Profile, and Validation workshops. These commands are used to create TOOL projects from the various iIS plans and to make iIS library distributions from these TOOL projects. As explained in [“About Registration” on page 137](#), it is these iIS library distributions that are registered with iIS process engines.

► **To make an iIS library distribution**

1. Use the `SetWorkspace` command to make a particular workspace current.
2. Use the `Open` command to open the current workspace.
3. Use the `ListPlans` command to display iIS plans in the workspace.
4. Use the `FindPlan` command to make a particular plan current.
5. Use the `CompilePlan` command to create a TOOL project for the plan.
6. Use the `MakeConductorDistribution` command to generate an iIS library distribution.

The library distribution is saved in the `FORTE_ROOT/appdist` directory on the central server node in your iIS system.

The commands needed to make iIS library distributions, except for `MakeConductorDistribution`, are documented in the *iPlanet UDS Fscript Reference Guide* and summarized in the table below:

Command	Arguments	Function
<code>Close</code>	—	Close the current workspace.
<code>IncludePublicPlan</code>	<i>plan_name</i>	Include a publicly-available plan in the current workspace.
<code>ListPublicPlans</code>	[<i>match_string</i>] [<i>show_unintegrated</i>] [<i>show_internal</i>]	List the publicly available plans in the repository.
<code>ListWorkspaces</code>	[<i>verbose_flag</i>]	List the workspaces in the repository.
<code>NewWorkspace</code>	<i>workspace_name</i>	Create a new workspace in current repository.
<code>Open</code>	[<i>permissions</i>] [<i>workspace_password</i>]	Open the workspace specified on the last <code>SetWorkspace</code> command.
<code>SetRepos</code>	<i>repository_name</i>	Set the name of the repository for the next <code>Open</code> command.
<code>SetWorkspace</code>	<i>workspace_name</i>	Set the name of the workspace for the next <code>Open</code> command.

Command	Arguments	Function
AddSupplierPlan	<i>plan_name</i>	Include a plan as a supplier plan to the current plan.
Commit	—	Commit changes to current workspace.
CompilePlan	[<i>force_flag</i>]	compile all out-of-date components in a plan.
ExportPlan	<i>file_name</i> [node ids]	Export all components of a plan to an export file.
FindPlan	<i>plan_name</i>	Make the specified plan current.
ImportPlan	<i>file_name</i>	Import a plan from the specified file.
IntegrateWorkspace	[<i>comment</i>] [<i>logfile_name</i>] [<i>baseline_password</i>]	Integrate the changes in this workspace into the shared repository.
ListPlans	[<i>match_string</i>] [*]	List the plans in the current workspace.
MakeConductor Distribution	—	Make an iIS library distribution from the current iIS plan. See page 292 .
RemoveSupplierPlan	<i>plan_name</i>	Remove a supplier plan from the current plan.
ShowPlan	—	Display information about the current plan.
ShowWorkspace	—	show the name of the current workspace and repository.
UpdateWorkspace	[<i>logfile_name</i>]	Update the current workspace with any changes in the system baseline since the last update.

Registering iIS Library Distributions

After iIS library distributions have been generated, they can be registered with iIS process engines. You can register any number of iIS distributions with an engine using Conductor Script, but you can register only one type of distribution (user profile, process definition, and so on) with one engine at a time. To register multiple distributions, you can write a script that performs all the registrations that you want.

► To register an iIS distribution with an engine

1. Use the `FindEngine` command to make a particular engine current.
2. Use the `ListConductorDistributions` command to display a list of iIS library distributions that are available for registration.
3. Use the appropriate `Register` command to register an available distribution of the corresponding type.

The commands needed to register iIS library distributions are documented in [Appendix A, “Conductor Script Commands”](#) and summarized in the table below:

Command	Arguments	Function	See...
<code>FindEngine</code>	<i>engine_name</i>	Set specified engine as the “current” engine. The engine (nor any of its components) does not have to be running.	page 284
<code>ListConductorDistributions</code>	[type]	Display a list of the specified type of iIS library distributions available for registration. May include process definitions, assignment rule dictionaries, user profiles, and user validations.	page 288
<code>ListEngines</code>	—	Display a list of configured iIS process engines.	page 289

Command	Arguments	Function	See...
RegisterAlias	<i>alias_name</i> <i>process_name</i> <i>engine_name</i> <i>environment</i>	Register the alias—representing a specified process running on a specified engine in a specified environment—with the current engine.	page 294
Register AssignmentRules	[<i>dictionary_name</i>]	Register all assignment rule dictionaries or the specified dictionary with the current engine.	page 294
Register ProcessDefinition	[<i>process_name</i>]	Register all process definitions or the specified process definition with the current engine.	page 295
Register UserProfile	[<i>user_profile_name</i>]	Register all user profiles or the specified user profile with the current engine.	page 295
RegisterValidation	<i>validation_name</i>	Register the specified validation with the current engine.	page 296

Unregistering iIS Library Distributions

Normally you unregister an iIS distribution from an engine if it is no longer needed. Cold-starting an engine unregisters all registrations; however, if you want to be more selective, you can unregister specific distributions using Conductor Script.

You can unregister any number of registered distributions, but you can only unregister one type of distribution (user profile, process definition, assignment rule dictionary...) from one engine at a time. To unregister multiple distributions, you can write a script that performs all the unregistrations that you want.

► **To unregister an iIS distribution from an engine**

1. Use the `FindEngine` command to make a particular engine current.
2. Use the `ListRegistrations` command to display a list of iIS distributions that are registered with the current engine.
3. Use the appropriate `UnRegister` command to unregister a distribution of the corresponding type.

The engine maintains only one validation, so registering a new validation implicitly unregisters the previous one.

The commands needed to unregister iIS distributions are documented in [Appendix A, “Conductor Script Commands”](#) and summarized in the table below:

Command	Arguments	Function	See...
<code>FindEngine</code>	<i>engine_name</i>	Set specified engine as the “current” engine. The engine (nor any of its components) does not have to be running.	page 284
<code>ListEngines</code>	—	Display a list of all running and configured iIS process engines in the environment.	page 289
<code>ListRegistrations</code>	[type]	List all registered process definitions, assignment rule dictionaries, user profiles, user validations and aliases (or just the specified type of distribution) for the current engine.	page 290
<code>Uninstall</code>	<i>name</i> <i>compatibility_level</i>	Deletes an iIS distribution of the specified name and level from the environment repository.	page 321
<code>UnRegisterAlias</code>	<i>alias_name</i>	Unregister the specified alias from the current engine.	page 322

Command	Arguments	Function	See...
UnRegister AssignmentRules	<i>dictionary_name</i> [<i>rule_name</i>]	Unregister all assignment rules (or just the specified assignment rule) in the specified assignment rule dictionary from the current engine.	page 322
UnRegister ProcessDefinition	<i>process_name</i>	Unregister the specified process definition from the current engine.	page 323
UnRegister UserProfile	<i>user_profile_name</i>	Unregister the specified user profile from the current engine.	page 324

Managing Process Execution with Conductor Script

This section describes how to perform some basic monitoring and management of process execution using Conductor Script. Normally these operations are more suited to iIS Console than to a command line utility. However, a number of Conductor Script commands are available to perform the following management tasks:

- monitoring and managing sessions
- monitoring and managing executing processes

For background information on iIS process execution, including how to perform process management using iIS Console, see [Chapter 6, “Managing Process Execution.”](#)

Monitoring and Managing Engine Sessions

This section describes how to perform a number of tasks regarding the monitoring and managing of sessions:

- obtaining state information about a session
- suspending or terminating sessions
- sending messages to sessions

► To obtain state information about a session

1. Use the `FindEngine` command to make a particular engine current
2. Use the `ListSessions` command to display a list of sessions with the current engine.
3. Use the `CreateFilter` command to filter the list of sessions, if desired.
4. Use the `ListSessions` command to display a list of filtered sessions.
5. Use the `ShowSession` command to display state information about a particular session.

► To suspend or terminate a session

1. Display a list of sessions as described in [Step 1](#) through [Step 4](#) above.
2. Enter `SuspendSession`, `TerminateSession`, `SuspendAllSessions`, or `TerminateAll Sessions` to suspend or terminate one or all sessions.

NOTE A session that has in-progress two-phase commit operations can be suspended, but cannot be terminated. For more information, see [“Monitoring and Managing Two-Phase Commit” on page 199](#).

► To send a message to one or more sessions

1. Use the `SendMessage` or `BroadcastMessage` command, depending on whether you want to send a message to one session or to all sessions.

The commands needed to monitor and manage iIS sessions are documented in [Appendix A, “Conductor Script Commands”](#) and summarized in the table below:

Command	Arguments	Function	See...
BroadcastMessage	<i>message_text</i> <i>priority</i>	Send a message with the specified priority to all sessions connected to the engine. The priority is a text string meaningful to the receiver.	page 275
FindEngine	<i>engine_name</i>	Set specified engine as the “current” engine. The engine (nor any of its components) does not have to be running.	page 284
ListEngines	—	Display a list of all running and configured iIS process engines in the environment.	page 289
ListSessions	—	List all sessions in the current engine.	page 291
SendMessage	<i>session_id</i> , <i>message_text</i> <i>priority</i>	Send a message with the specified priority to the specified session. The priority is a text string meaningful to the receiver.	page 299
ShowSession	<i>session_id</i>	Show details of the given session, including the activity list.	page 308
SuspendAllSessions	—	Suspend all active sessions in the current engine.	page 319
SuspendSession	<i>session_id</i>	Suspend the specified active session.	page 320
TerminateAllSessions		Terminate all sessions in the current engine.	page 320
TerminateSession	<i>session_id</i>	Terminate the specified session.	page 321

Monitoring and Managing Process Execution

There are a number of situations where you are likely to want to monitor or intervene in process execution to resolve problems. Depending on the situation, you might wish to take some of the actions described in the following paragraphs.

Managing Process Instances

➤ **To check the status of a process instance**

1. Use the `FindEngine` command to make a particular engine current
2. Use the `ListProcesses` command to display a list of process instances in the current engine.
3. Use the `CreateFilter` command to filter the list of process instances, if desired.
4. Use the `ListProcesses` command to display a list of filtered process instances.
5. Use the `ShowProcess` command to display information about a particular process instance.

The output shows information about the activities, timers, and process attributes of the process instance (see [Figure 6-2 on page 180](#)). You can get further information about these objects using their corresponding `Show` command.

➤ **To abort a process instance**

1. Display a list of process instances as described in [Step 1](#) through [Step 4](#) above.
2. Use the `AbortProcess` or `AbortAllProcesses` command, depending on whether you wish to abort a specific process instance or all process instances.

Managing Activity Instances

➤ **To check the status of an activity instance**

1. Use the `FindEngine` command to make a particular engine current
2. Use the `ListActivities` command to display a list of process instances in the current engine.
3. Use the `CreateFilter` command to filter the list of activities, if desired.
4. Use the `ListActivities` command to display a list of filtered activities.

5. Use the `ShowActivity` command to display information about a particular activity.

The output shows information about the activity, its attributes and its linked timers. You can get further information about these objects using their corresponding `Show` command.

➤ **To change the state of an activity**

1. Display a list of activities as described in [Step 1](#) through [Step 4](#) above.
2. Use the `ShowActivity` command to display information about a particular activity.
3. Use the `AbortActivity`, `ReadyActivity`, or `RollbackActivity` command to abort an activity, change its state from `PENDING` to `READY`, or change its state from `ACTIVE` to `READY`.

Managing Activity Queues

➤ **To list the contents of an activity queue**

1. Use the `FindEngine` command to make a particular engine current.
2. Use the `ListActivityQueues` command to display a list of queues in the current engine.
3. Use the `CreateFilter` command to filter the list of queues, if desired.
4. Use the `ShowActivityQueue` command to display a list of activities in a specified queue.

This command displays information about the activities in the queue. You can get further information about these activities using the `ShowActivity` command.

➤ **To reprioritize an activity in a queue**

1. Display a list of activities in a queue as described in [Step 1](#) through [Step 4](#) above.
2. Use the `ShowActivity` command to display information about a particular activity.
3. Use the `SetQueuedActivityPriority` command to change the value of the queue prioritizing process attribute for the activity. The engine reorders the queue.

Managing Timers

► To check the status of a timer

1. Use the `FindEngine` command to make a particular engine current.
2. Use the `ListTimers` command to display a list of timers in the current engine.
3. Use the `CreateFilter` command to filter the list of timers, if desired.
4. Use the `ListTimers` command to display a list of filtered timers.
5. Use the `ShowTimer` command to display information about a particular timer.

The output shows information about the timer, its state and its expiration time.

► To change the state of a timer or change its expiration time

1. Display a list of timers as described in [Step 1](#) through [Step 4](#) above.
2. Use the `ShowTimer` command to display information about a particular timer.
3. Use the `SetTimer`, `SetTimerDeadline`, or `SetTimerElapsed` command to turn a timer on or off, or to change the expiration date or time, depending on whether the timer is a deadline timer or an elapsed timer.

Managing Process Attributes

► To check the value or lock state of a process attribute

1. Use the `FindEngine` command to make a particular engine current.
2. Use the `ListProcesses` command to display a list of process instances in the current engine.
3. Use the `CreateFilter` command to filter the list of process instances, if desired.
4. Use the `ListProcesses` command to display a list of filtered process instances.
5. Use the `ShowProcess` command to display information about a particular process instance.

The output shows information about the process attributes of the process instance.

- **To change the value of a process attribute**
 1. Display a list of attributes as described in [Step 1](#) through [Step 4](#) above.
 2. Use the `SetAttributeValue` command to set the value of any listed process attribute.

- **To remove an attribute lock**
 1. Display a list of attributes as described in [Step 1](#) through [Step 4](#) above.
 2. Use the `RemoveReadLock` command to remove a read lock, or the `RemoveWriteLock` command to remove a write lock.

Checking for Bottlenecks in Process Execution

- **To find a process bottleneck**
 1. Use the `FindEngine` command to make a particular engine current.
 2. Use the `ListActivities` command to display a list of activity instances in the current engine.
 3. Use the `CreateFilter` command to filter the list of activity instances by process name.
 4. Use the `ListActivities` command to display a list of filtered activity instances.
 5. Look for an activity name with more than an average number of instances.
 6. Use the `CreateFilter` command to filter the list of activity instances by the suspected bottleneck activity name.
 7. Use the `ListActivities` command to display a list of filtered activity instances.
 8. Check the state of activities in the listing.
 9. Use the `ShowActivity` command to display more information about each particular activity instance to determine what is causing the bottleneck.

The commands needed to intervene in execution of an iIS process instance are documented in [Appendix A, “Conductor Script Commands”](#) and summarized in the table below:

Command	Arguments	Function	See...
AbortActivity	<i>process_id</i> <i>activity_name</i>	Place the specified activity in the ABORTED state.	page 273
AbortAllProcesses	—	Abort all process instances in the current engine.	page 274
AbortProcess	<i>process_id</i>	Abort the specified process instance.	page 274
CompleteActivity	<i>process_id</i> <i>activity_name</i>	Change the state of the specified activity from ACTIVE to COMPLETED.	page 276
ConsultActivity	<i>process_id</i> , <i>activity_id</i> <i>state</i> <i>consultation_rule</i> <i>return_rule</i> [<i>user_name1</i>] [<i>other_info1</i>] [<i>user_name2</i>] [<i>other_info2</i>]	Delegate the specified activity (in the specified state) to consultant users using the specified consultation assignment rule. A consultant session returns the activity using the return_rule assignment rule. The user name and other info strings are supplied if required by the consultation and return assignment rules.	page 276
CreateActivity	<i>process_id</i> , <i>activity_name</i>	Create an activity of the specified name in the specified process instance and place it in the PENDING state.	page 278
CreateFilter	<i>time_interval</i> <i>filter_expression</i>	Create a filter for events posted on the current engine object which conform to the specified filter expression.	page 279
DelegateActivity	<i>process_id</i> , <i>activity_id</i> <i>state</i> <i>delegation_rule</i> [<i>user_name</i>] [<i>other_info</i>]	Delegate the specified offered activity (in the specified state) to other users using the specified delegation assignment rule. The user name and other info string are supplied if required by the delegation rule.	page 281
DeleteFilter	<i>filter_id</i>	Delete the specified filter.	page 283
ListActivities	—	List all activities in the current engine.	page 287
ListActivityQueues	[<i>process_name</i>]	Display activity queues for the specified process (or all activity queues) in the current engine.	page 287

Command	Arguments	Function	See...
ListFilters	—	List all existing event filters for the current engine.	page 289
ListProcesses	[short] , [name process_name]	List process instances for a specified process (or all process instances) in the current engine, or list a summary of process instances by process name.	page 289
ListTimers	—	List all timers in the current engine.	page 291
ReadyActivity	process_id, activity_name	Change the state of the specified activity from PENDING to READY.	page 293
RemoveReadLock	process_id attribute_name	Remove a shared lock on the specified process attribute.	page 297
RemoveWriteLock	process_id attribute_name	Remove an exclusive lock on the specified process attribute.	page 297
RollbackActivity	process_id activity_name	Change the state of the specified activity from ACTIVE to READY.	page 298
SetAttributeValue	process_id attribute_name attribute_type value	Set the value of the specified attribute for a specified process instance.	page 300
SetQueuedActivity Priority	process_id activity_id prioritizing_value	Reprioritize activities in an activity queue by setting the prioritizing process attribute for a specified activity to a specified value.	page 301
SetTimer	process_id timer_name state [reset_control]	Set the specified timer to the specified state (ON, OFF, or no change) and resume operation (or reset the timer)	page 302
SetTimerDeadline	process_id timer_name state expiration_time	Set the specified deadline timer to the specified state (ON, OFF, or no change) and set a new expiration time.	page 303
SetTimerElapsed	process_id timer_name state time_interval	Set the specified elapsed timer to the specified state (ON, OFF, or no change) and set a new elapsed time interval until expiration.	page 304
ShowActivity	process_id, activity_name	Show details of the specified activity.	page 304

Command	Arguments	Function	See...
ShowActivityQueue	<i>process_name</i> , [<i>queue_name</i>]	Display the contents of the specified queue (or all queues) for the specified process in the current engine.	page 305
ShowProcess	<i>process_id</i>	Show details of specified process instance, including all activities and the values of all process attributes.	page 307
ShowTimer	<i>process_id</i> <i>timer_name</i>	Show details of the specified timer.	page 313
StartActivity	<i>process_id</i> , <i>activity_name</i> <i>session_id</i>	Change the state of the specified activity from READY to ACTIVE.	page 314
StartTimer	<i>process_id</i> <i>timer_name</i>	Start the specified timer. Place it in the ON state.	page 317
StopTimer	<i>process_id</i> <i>timer_name</i>	Stop the specified timer. Place it in the OFF state.	page 319

Monitoring and Managing Two-Phase Commit Transactions

The iIS two-phase commit protocol facilitates the synchronization of iIS process engine operations with application transactions by making it possible to place iIS engine operations in a transactional context. For information on this capability, see [“Monitoring and Managing Two-Phase Commit” on page 199](#).

As a system manager, you can monitor and manage these iIS transactions, as described in the following sections.

Monitoring Two-Phase Commit Operations

You can use Conductor Script commands to monitor the status of in-progress iIS engine operations, as indicated in the table below. These commands print out the transaction ID of any iIS operations that are in a PREPARE phase.

Command	Arguments	Function	See...
ShowActivity	<i>process_id</i> <i>activity_id</i>	Indicates any in-progress two-phase commit transactions in the activity.	page 304

Command	Arguments	Function	See...
ShowProcess	<i>process_id</i>	Indicates any in-progress two-phase commit transactions in the process.	page 307
ShowSession	<i>session_id</i>	Indicates whether the session has two-phase commit mode enabled, and if so, any in-progress transaction. For example, it may indicate the following: Two phase commit mode is enabled. There is a client operation "CompleteActivity" in PREPARE phase.	page 308

Managing Two-Phase Commit Operations

If a session should be suspended for any of a number of reasons, any iIS engine transaction in a PREPARE phase is retained on the session, awaiting resolution (commit or rollback). The process client application is normally responsible for resolving these transactions; however, in the case of failure, these transactions may be left permanently in the PREPARE phase. Because of this, it is up to a system manager to check for and resolve any transactions left in a PREPARE phase.

For example, a client application may perform an activity, but fail before it can notify the engine to commit the CompleteActivity operation. Similarly, the engine could fail before receiving the commit. Both these situations result in inconsistency of state information between the client application and engine.

To properly resolve iIS engine transactions in a PREPARE phase, however, you must investigate whether related application transactions were committed or aborted. How you do this depends on the details of the application and how it keeps track of transaction IDs.

You can use the following Conductor Script commands to identify and resolve transactions in a PREPARE phase:

Command	Argument	Function	See...
CommitTransaction	<i>session_ID</i> <i>transaction_ID</i>	Commits the in-progress iIS engine transaction for the specified session or transaction ID.	page 275

Command	Argument	Function	See...
ListTransactions		Lists information for all in-progress iIS engine transactions (that is, those in a PREPARE phase) for all sessions in the current engine.	page 291
RollbackTransaction	<i>session_ID</i> <i>transaction_ID</i>	Rolls back the in-progress iIS transaction for the specified session or transaction ID.	page 298

➤ **To resolve transactions after a client or engine failure**

1. Start Conductor Script from a command line and invoke ListTransactions.

```
cscript >ListTransactions
```

The output of this command identifies all sessions, and their respective status, that have transactions in a PREPARE phase. For example, if a CreateProcess transaction had been prepared before the client crashed and there was only one session in progress, you would see output such as the following:

```
Session Name:s1 Id:200 Status:Suspended
TransactionId:My_Transaction_Id1 Current operation:
"CreateProcess"
```

2. Commit or roll back the prepared transaction:

```
cscript >CommitTransaction 200
```

3. Verify the commit or rollback of the transaction:

```
cscript >ShowSession 200
```

The output of the ShowSession command should indicate that the transaction has been resolved.

Conductor Script Commands

This appendix is an alphabetically ordered reference of the Conductor Script commands.

Conductor Script is a command line utility built on top of the iPlanet UDS Fscript command utility. As such, it includes Fscript commands for performing repository-based operations, as well as a large number of iIS-specific commands.

This appendix documents only the iIS-specific Conductor Script commands. For documentation of Fscript Commands accessible through Conductor Script, see the *Fscript Reference Guide*.

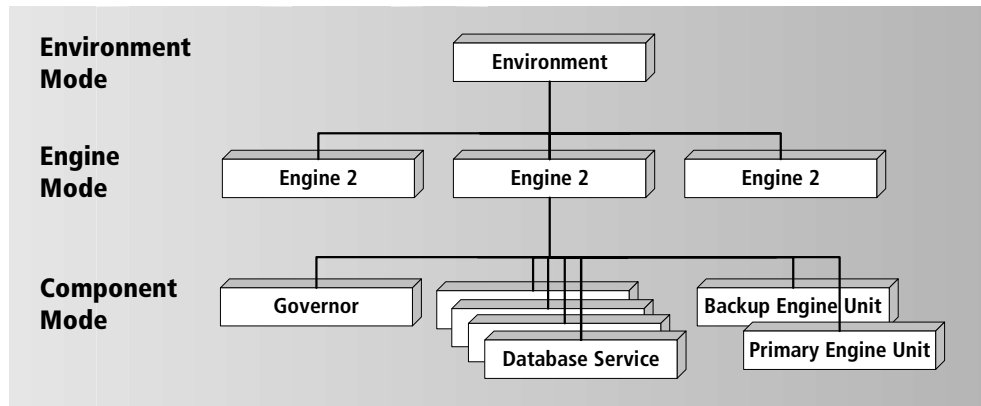
An introduction summarizes and groups the Conductor Script commands according to their function.

Conductor Script Command Summary

The following tables summarize Conductor Script commands and group them into the following categories, each of which represents a level within the hierarchy shown in [Figure A-1](#):

- environment mode commands
- engine mode commands
- component mode commands

Figure A-1 Hierarchy of Conductor Script Levels



Each Conductor Script command is defined on one of these levels. To use a command, you navigate the hierarchy, making either an engine or an engine component “current,” and then invoke the command.

NOTE Conductor Script also provides access to all iPlanet UDS Fscript commands (except for the Run, RunDistrib, and RunFile commands). Fscript is an iPlanet UDS command line interface that lets you create new plans and projects, examine components, define and modify classes, partition, test, and run distributed applications, and define and deploy libraries. For information on how to use Fscript commands for writing and running Conductor Script scripts, see [Chapter 8, “Using the Conductor Script Utility.”](#) For more complete information on Fscript commands, see the iPlanet UDS *Fscript Reference Guide*.

Environment Mode Commands

The Conductor Script environment mode commands work only in the active environment. They are used mostly to specify the current engine and current node, and also to make library distributions that can be registered with iIS process engines.

Command	Arguments	Function	See...
FindEngine	<i>engine_name</i>	Set specified engine as the “current” engine. The engine (nor any of its components) does not have to be running.	page 284
IIOPServer	[start stop show]	Starts, stops, or displays information about the IIOP service needed to support CORBA/IIOP iIS process client applications.	page 287
ListConductor Distributions	[type]	Display a list of the specified type of iIS library distributions available for registration. May include process definitions, assignment rule dictionaries, user profiles, and user validations.	page 288
ListEngines	—	Display a list of all running and configured iIS process engines in the environment.	page 288
MakeConductor Distribution	—	Generate a library distribution for the current iIS plan. Plan must be created using the iIS process development workshops. (Automatically boosts compatibility level of distribution.)	page 292

Command	Arguments	Function	See...
SetPassword	[old_password] [new_password]	Set password protection on opening Conductor Script in your active environment. Once you set a password, other users may not open Conductor Script (or iIS Console, or iPlanet UDS Environment Console, or Escript) in that environment without using the password.	page 300
ShowEngine	<i>engine_name</i>	Display status of the specified engine.	page 306
Uninstall	<i>name</i> <i>compatibility_level</i>	Deletes an iIS distribution of the specified name and compatibility level from the environment repository. Normally performed after unregistering an iIS distribution from all engines in the environment.	page 321

Engine Mode Commands: Engine Management

The Conductor Script engine mode commands work only for the current engine. They are used mostly to start components and specify the current component.

Command	Arguments	Function	See...
FindDBService	<i>service_name</i>	Set the specified database service for the current engine as the current component. The database service must be running.	page 283
FindGovernor	—	Set the governor for the current engine as the current component. The governor must be running.	page 284

Command	Arguments	Function	See...
FindNode	<i>node_name</i>	Set the specified node as the “current” node. The node must exist in the environment.	page 285
FindPrimary	—	Set the primary engine unit for the current engine as the current component. The primary unit must be ONLINE.	page 285
FindUnit	<i>unit_name</i>	Set the specified engine unit for the current engine as the current component. The engine unit must be running.	page 286
SetPrimary	<i>unit_name</i>	Set the specified engine unit for the current engine as the primary unit. This will change the states of the engine units to make the specified unit ONLINE.	page 301
ShowConfiguration	—	Display contents of the configuration file of the current engine. (No engine components need be running for this command to work.)	page 309
ShowStatus	—	Display status of the current engine. (No engine components need be running for this command to work.)	page 309
Shutdown	—	Shut down the current engine. This shuts down the governor, all database services, and all engine units.	page 314
StartDBService	<i>service_name</i> <i>priority</i>	Start the specified database service for the current engine on the current node.	page 315

Command	Arguments	Function	See...
StartEngine	[newLog] [newState] [newRegistration] [cold]	Start all components of the engine as specified in the engine configuration file. (Includes various start options that create new engine database tables.)	page 315
StartGovernor	—	Start the governor for the current engine on the current node.	page 316
StartUnit	<i>unit_name</i> [cold]	Start the specified engine unit for the current engine on the current node. (Includes cold start option to create engine database.)	page 317
WaitForStartup	<i>unit_name</i> <i>timeout</i>	Wait the specified timeout period (in seconds) for the specified engine unit to start up, and prints an error message if engine unit does not start within specified time. Used for writing automated scripts.	page 324

Engine Mode Commands: Process Execution Management

There are a large number of Conductor Script commands that let you monitor and manage sessions, process execution, and registration—functions performed by the primary engine unit. These commands, nevertheless, are available in engine mode—when the engine is current. The Conductor Script commands for managing process execution are also available when the primary engine unit is current.

Command	Arguments	Function	See...
AbortActivity	<i>process_id</i> , <i>activity_name</i>	Place the specified activity in the ABORTED state.	page 273
AbortAllProcesses	—	Abort all process instances in the current engine.	page 274
AbortProcess	<i>process_id</i>	Abort the specified process instance.	page 274
BroadcastMessage	<i>message_text</i> , <i>priority</i>	Send a message with the specified priority to all sessions connected to the engine. The priority is a text string meaningful to the receiver.	page 275
CommitTransaction	<i>session_ID</i> <i>transaction_ID</i>	Commits the in-progress iIS transaction for the specified session or transaction ID.	page 275
CompleteActivity	<i>process_id</i> , <i>activity_name</i>	Change the state of the specified activity from ACTIVE to COMPLETED.	page 276
ConsultActivity	<i>process_id</i> , <i>activity_id</i> <i>state</i> <i>consultation_rule</i> <i>return_rule</i> [<i>user_name1</i>] [<i>other_info1</i>] [<i>user_name2</i>] [<i>other_info2</i>]	Delegate the specified activity (in the specified state) to consultant users using the specified consultation assignment rule. A consultant session returns the activity using the return_rule assignment rule. The user name and other info strings are supplied if required by the consultation and return assignment rules.	page 276
CreateActivity	<i>process_id</i> , <i>activity_name</i>	Create an activity of the specified name in the specified process instance and place it in the PENDING state.	page 276
CreateFilter	<i>time_interval</i> <i>filter_expression</i>	Create a filter for events posted on the current engine object which conform to the specified filter expression.	page 279

Command	Arguments	Function	See...
DelegateActivity	<i>process_id</i> , <i>activity_id</i> <i>state</i> <i>delegation_rule</i> [<i>user_name</i>] [<i>other_info</i>]	Delegate the specified offered activity (in the specified state) to other users using the specified delegation assignment rule. The user name and other info string are supplied if required by the delegation rule.	page 281
DeleteFilter	<i>filter_id</i>	Delete the specified filter.	page 281
FlushLog	—	Remove all the data held in the history log database tables.	page 286
ListActivities	—	List all activities in the current engine.	page 287
ListActivityQueues	[<i>process_name</i>]	Display activity queues for the specified process (or all processes) in the current engine.	page 287
ListFilters	—	List all existing event filters for the current engine.	page 289
ListProcesses	[<i>short</i>], [<i>name process_name</i>]	List process instances for a specified process (or list all process instances) in the current engine, or list a summary of process instances by process name.	page 289
ListRegistrations	[<i>type</i>]	List all registered process definitions, assignment rule dictionaries, user profiles, user validations and aliases (or just the specified type of distribution) for the current engine.	page 290
ListSessions	—	List all sessions in the current engine.	page 291
ListTimers	—	List all timers in the current engine.	page 291
ListTransactions		Lists information for all in-progress iIS engine transactions (that is, those in a PREPARE phase) for all sessions in the current engine.	page 291

Command	Arguments	Function	See...
ReadyActivity	<i>process_id</i> , <i>activity_name</i>	Change the state of the specified activity from PENDING to READY.	page 293
RegisterAlias	<i>alias_name</i> , <i>process_name</i> , <i>engine_name</i> , <i>environment</i>	Register the alias—representing a specified process running on a specified engine in a specified environment—with the current engine.	page 294
Register AssignmentRules	[<i>dictionary_name</i>]	Register all assignment rule dictionaries or the specified assignment rule dictionary with the current engine.	page 294
Register ProcessDefinition	[<i>process_name</i>]	Register all process definitions or the specified process definition with the current engine.	page 295
Register UserProfile	[<i>user_profile_name</i>]	Register all user profiles or the specified user profile with the current engine.	page 295
RegisterValidation	<i>validation_name</i>	Register the specified validation with the current engine.	page 296
RemoveReadLock	<i>process_id</i> , <i>attribute_name</i>	Remove a shared lock on the specified process attribute.	page 297
RemoveWriteLock	<i>process_id</i> , <i>attribute_name</i>	Remove an exclusive lock on the specified process attribute.	page 297
RollbackActivity	<i>process_id</i> , <i>activity_name</i>	Change the state of the specified activity from ACTIVE to READY.	page 298
RollbackTransaction	<i>session_ID</i> <i>transaction_ID</i>	Rolls back the in-progress iIS transaction for the specified session or transaction ID.	page 298
SendMessage	<i>session_id</i> , <i>message_text</i> , <i>priority</i>	Send a message with the specified priority to the specified session. The priority is a text string meaningful to the receiver.	page 299
SetAttributeValue	<i>process_id</i> <i>attribute_name</i> <i>attribute_type</i> <i>value</i>	Set the value of the specified attribute for a specified process instance.	page 300

Command	Arguments	Function	See...
SetQueuedActivity Priority	<i>process_id</i> , <i>activity_id</i> <i>prioritizing_value</i>	Reprioritize activities in an activity queue by setting the prioritizing process attribute for a specified activity to a specified value.	page 301
SetTimer	<i>process_id</i> , <i>timer_name</i> <i>state</i> [reset]	Set the specified timer to the specified state (ON, OFF, or no change) and resume operation (or reset the timer)	page 302
SetTimerDeadline	<i>process_id</i> , <i>timer_name</i> <i>state</i> <i>expiration_time</i>	Set the specified deadline timer to the specified state (ON, OFF, or no change) and set a new expiration time.	page 303
SetTimerElapsed	<i>process_id</i> , <i>timer_name</i> <i>state</i> <i>time_interval</i>	Set the specified elapsed timer to the specified state (ON, OFF, or no change) and set a new elapsed time interval until expiration.	page 304
ShowActivity	<i>process_id</i> , <i>activity_name</i>	Show details of the specified activity.	page 302
ShowActivityQueue	<i>process_name</i> , [<i>queue_name</i>]	Display the contents of the specified queue (or all queues) for the specified process in the current engine.	page 305
ShowProcess	<i>process_id</i>	Show details of specified process instance, including all activities and the values of all process attributes.	page 307
ShowSession	<i>session_id</i>	Show details of the given session, including the activity list.	page 308
ShowTimer	<i>process_id</i> , <i>timer_name</i>	Show details of the specified timer.	page 313
StartActivity	<i>process_id</i> , <i>activity_name</i> <i>session_id</i>	Change the state of the specified activity from READY to ACTIVE.	page 314
StartTimer	<i>process_id</i> , <i>timer_name</i>	Start the specified timer. Place it in the ON state.	page 317
StopTimer	<i>process_id</i> , <i>timer_name</i>	Stop the specified timer. Place it in the OFF state.	page 319

Command	Arguments	Function	See...
SuspendAllSessions	—	Suspend all active sessions in the current engine.	page 319
SuspendSession	<i>session_id</i>	Suspend the specified active session.	page 320
TerminateAllSessions		Terminate all sessions in the current engine.	page 320
TerminateSession	<i>session_id</i>	Terminate the specified session.	page 321
UnRegisterAlias	<i>alias_name</i>	Unregister the specified alias from the current engine.	page 322
UnRegister AssignmentRules	<i>dictionary_</i> <i>name</i> [<i>rule_name</i>]	Unregister all assignment rules (or just the specified assignment rule) in the specified assignment rule dictionary from the current engine.	page 322
UnRegister ProcessDefinition	<i>process_name</i>	Unregister the specified process definition from the current engine.	page 324
UnRegister UserProfile	<i>user_profile_</i> <i>name</i>	Unregister the specified user profile from the current engine.	page 324

Component Mode Commands

The Conductor Script engine component mode commands work only for the current component.

Generic component

A few commands work for any current component.

Command	Parameters	Function	See...
ShowStatus	[short]	Display the status of the current component. If the current component is an engine unit in the ONLINE state, then this command shows the full instrumentation implemented for all the engine's internal manager objects.	page 309
ShowLogFlags	—	Display the logger message filters (logger flags) set for the current component (partition logger settings).	page 307
ModLogFlags	+ (message_filters) - (message_filters)	modify the logger message filters (logger flag) set for the current component.	page 292
Shutdown	—	Shut down the current component.	page 314
FindParentEngine	—	Make the component's parent engine current. Puts you in "engine mode."	page 285

Engine Unit

One command applies only if the current component is an engine unit.

Command	Parameters	Function	See...
SetState	state	Set the state for the current engine unit to ONLINE or STANDBY.	page 302

Conductor Script Commands

The following is a listing of all Conductor Script commands.

AbortActivity

The `AbortActivity` command aborts the specified activity, placing it in the `ABORTED` state.

```
AbortActivity process_id activity_name
```

Argument	Description
<i>process_id</i>	The process instance in which the activity was created.
<i>activity_name</i>	The name of the activity.

The `AbortActivity` command is only available when an engine is current. Use the `FindEngine` Script command to set the current engine.

When you issue the `AbortActivity` command, the engine performs the following actions:

- discards pending process attribute updates—rolls back changes that were made in performing the activity and frees locks on process attributes associated with the activity
- evaluates the activity's `OnAbort` method, if one is specified in the process definition
- evaluates the activity's `OnAbort` routing methods—if none are specified, the engine by default aborts the process
- checks for a process stall condition—when all other activity instances are either `COMPLETED` or `PENDING` (no activities are in `ACTIVE` or `READY` state) and no timers are active—and aborts the process if a stall condition is detected.
- deletes the `ABORTED` activity

AbortAllProcesses

The `AbortAllProcesses` command aborts all process instances for the current engine.

`AbortAllProcesses`

The `AbortAllProcesses` command is only available for the current engine. Use the `FindEngine` Script command to set the current engine and the `ListProcesses` Script command to display a list of process instances before aborting them.

When you issue the `AbortAllProcesses` command, the engine posts an alarm for each process instance to be aborted.

AbortProcess

The `AbortProcess` command aborts a specified process instance for the current engine.

`AbortProcess process_id [process_id...]`

Argument	Description
<code>process_id</code>	The specified process instance. (A list of up to ten <code>process_ids</code> separated by spaces is supported.)

The `AbortProcess` command is only available for the current engine. Use the `FindEngine` Script command to set the current engine and the `ListProcesses` Script command to display a list of process instances before selecting the process to abort.

When you issue the `AbortProcess` command, the engine posts an alarm for the process instance to be aborted.

BroadcastMessage

The `BroadcastMessage` command sends a message to all active sessions on the current engine.

`BroadcastMessage message urgency`

Argument	Description
<code>message</code>	An alphanumeric string that constitutes the text of the message. Message text must be enclosed in double quotes.
<code>urgency</code>	One of two specified string constants: INFORMATIONAL or CRITICAL.

The `BroadcastMessage` command is only available when an engine is current. Use the `FindEngine` Script command to set the current engine.

The `BroadcastMessage` command is used to notify all sessions about a pending administrative action, such as changing the state of the primary engine from ONLINE to STANDBY.

CommitTransaction

The `CommitTransaction` command commits an in-progress iIS engine transaction (that is, one in a PREPARE phase). For details on two-phase commit operations, see [“Monitoring and Managing Two-Phase Commit” on page 199](#).

`CommitTransaction {session_ID | transaction_ID}`

Argument	Description
<code>session_ID</code>	The identifier of the session which has a transaction in a PREPARE phase. A session can have only one in-progress two-phase transaction.
<code>transaction_ID</code>	The identifier of the transaction in a PREPARE phase.

To roll back an in-progress iIS engine transaction, use the `RollbackTransaction` command. To list all in-progress iIS transactions for a given engine, use the `ListTransactions` command.

CompleteActivity

The `CompleteActivity` command completes the specified activity, putting it in `COMPLETED` state.

```
CompleteActivity process_id activity_name
```

Argument	Description
<code>process_id</code>	The process instance in which the activity was created.
<code>activity_name</code>	The name of the activity.

The `CompleteActivity` command is only available when an engine is current. Use the `FindEngine` Script command to set the current engine.

When you issue the `CompleteActivity` command, the engine performs the following actions:

- commits pending process attribute updates—saves changes that were made in performing the activity and frees locks on process attributes associated with the activity
- evaluates the activity's `OnComplete` method, if one is specified
- evaluates the activity's `OnComplete` router methods
- checks for a process stall condition—when all other activity instances are either `COMPLETED` or `PENDING` (no activities are in `ACTIVE` or `READY` state) and no timers are active—and aborts the process if a stall condition is detected
- deletes the `COMPLETED` activity

ConsultActivity

The `ConsultActivity` command hands off the specified offered activity to other users using the specified consultation assignment rule. A consulted session returns the activity using an assignment rule specified by the `return_rule`.

```
ConsultActivity process_id activity_id state consultation_rule
return_rule [user_name1 [other_info1] [user_name2 [other_info2]]]
```

Argument	Description
<i>process_id</i>	The process instance in which the activity was created.
<i>activity_id</i>	The activity instance to be delegated.
<i>state</i>	The state of the activity instance to be delegated (READY or ACTIVE).
<i>consultation_rule</i>	The assignment rule used to hand off the activity to consulted sessions.
<i>return_rule</i>	The assignment rule used to return the activity to delegating users.
<i>user_name1</i>	The user name, if any, required by the <i>consultation_rule</i> . If this argument is not required, put a null argument enclosed in double quotes ("").
<i>other_info1</i>	Otherinfo string of the user name passed to the <i>consultation_rule</i> , if any. If this argument is not required, put a null argument enclosed in double quotes ("").
<i>user_name2</i>	The user name if any, required by the <i>return_rule</i> .
<i>other_info2</i>	Otherinfo string of the user name passed to the <i>return_rule</i> , if any.

The `ConsultActivity` command is only available when an engine is current. Use the `FindEngine` Script command to set the current engine.

The `ConsultActivity` command is used to hand off the work associated with an offered activity to someone else, one or more consulted users, who then return it to one or more originating users. The activity must be in READY or ACTIVE state before the `ConsultActivity` command is used. If the activity is in an ACTIVE state, any open attributes are updated as necessary, the accessor is closed, and the activity's state is changed to READY. The activity is offered to sessions as permitted by the consultation assignment rule.

NOTE The *state* parameter is used to confirm that the current state of the activity is what you expect when you issue the `ConsultActivity` command. If the state has changed, the engine does not execute the command.

The `user_name1` and `other_info1` arguments must be supplied if required by the consultation assignment rule. For example, if a consultation rule named “ManagerOf” looks for the manager of the user who is handing off to a consultant, then this user’s user name and other info (in this case the manager’s name) would be passed to the ManagerOf assignment rule. This behavior is similar in function to the linked activity mechanism described in the *iIS Process Development Guide*.

Once the consulted user has completed work on the activity, the activity is offered back according to the assignment rule specified by `return_rule`. The `user_name2` and `other_info2` arguments must be supplied if required by the `return_rule`. For example, if a `return_rule` assignment rule named “SameAs” looks for the original user who passed off the activity to a consultant, then that original user’s name would be passed to the `return_rule` assignment rule.

Once the user to whom the activity is returned has completed (or aborted) the activity, process execution continues as specified in the process definition.

CreateActivity

The `CreateActivity` command creates the specified activity, placing it in PENDING state.

```
CreateActivity process_id activity_name
```

Argument	Description
<code>process_id</code>	The process instance in which the activity is to be created.
<code>activity_name</code>	The name of the activity.

The `CreateActivity` command is only available when an engine is current. Use the `FindEngine` Script command to set the current engine.

When the `CreateActivity` command creates an activity, it is placed in PENDING state. Its `Trigger` method is evaluated each time the `router` method of a preceding activity is evaluated as TRUE or when a process attribute changes value. When the trigger conditions are fulfilled, the activity can transition into READY state. To override the trigger conditions, you can use the `ReadyActivity` Script command after creating the activity.

CreateFilter

The `CreateFilter` command creates a filter for events posted on the current engine object.

```
CreateFilter time_interval filter_expression
```

Argument	Description
<i>time_interval</i>	The time during which the filter batches up events before reporting them, in milliseconds.
<i>filter_expression</i>	Expression specifying filter criteria.

The `CreateFilter` command is only available when an engine is current. Use the `FindEngine` Script command to set the current engine.

The `CreateFilter` command assigns a filter ID to the new filter.

The *filter_expression* argument provides the criteria used to filter the universe of events posted by the engine. These events include all state changes to sessions, process instances, and activities, as well as changes in process attributes. The filter expression uses a number of comparison operators to specify three filter expression elements. The three elements—type, object class, and object instance—and the operators are described below:

Event Type

Type	Value	Description
Object creation	1	Events reporting creation of sessions, process instances, or activities
Object deletion	2	Events reporting deletion of sessions, process instances, or activities
Attribute change	3	Events reporting changes in process attribute values
Alarm report	4	Events of a particularly important nature regarding process execution

Object Class

Class	Value	Description
Session	1	Events involving sessions
Process instance	2	Events involving process instances
Activity	3	Events involving activities
Engine	4	Events involving engine objects
Timer	5	Events involving timers
Registration	6	Events involving registrations

Object Instance

This element specifies a particular instance identification, if desired—for example, a session name, activity name, or process ID.

Class	Instance Identification
Session	session_id
Process instance	process_id
Activity	process_id and activity_id

Operators

The table below shows the different operators that can be used with the filter elements to specify the filter criteria. The two unary operators at the top of the table have higher priority than the binary operators that follow, which have higher priority than the logical operators at the bottom of the table.

Operator	Meaning
?	instance identification present (in the event)
~	instance identification not present (in the event)
=	equal
<>	not equal
<	less than

Operator	Meaning
>	greater than
<=	less than or equal to
>=	greater than or equal to
and	logical and
or	logical or
not	logical not

Examples

Some example filter expressions follow:

Filter Expression	Meaning
class = 1 and name = "Session1"	Filter all events from the session whose name is "Session1".
type = 1 and class = 2	Filter all process creations.
type = 4	Filter all alarms.
class = 3 and process_id = 45	Filter all events from the activities of process 45.

When the `CreateFilter` command sets up the filter as specified, events that satisfy the filter expression are printed to the screen.

DelegateActivity

The `DelegateActivity` command delegates the specified offered activity to other users using the specified delegation assignment rule.

```
DelegateActivity process_id activity_id state delegation_rule
[user_name [other_info]]
```

Argument	Description
<i>process_id</i>	The process instance in which the activity was created.
<i>activity_id</i>	The activity instance to be delegated.

Argument	Description
<i>state</i>	The state of the activity instance to be delegated (READY or ACTIVE).
<i>delegation_rule</i>	The assignment rule used to assign the activity to sessions, that is, to delegated users.
<i>user_name</i>	The user name, if any, required by the <i>delegation_rule</i> . If this argument is not required, place a null argument in quotation marks ("").
<i>other_info</i>	Otherinfo string of the user name passed to the <i>delegation_rule</i> , if any.

The `DelegateActivity` command is only available when an engine is current. Use the `FindEngine` Script command to set the current engine.

The `DelegateActivity` command is used to hand off the work associated with an offered activity to someone else: one or more delegated users. The delegated activity must be in READY or ACTIVE state before the `DelegateActivity` command is used. If the activity is in ACTIVE state, any open attributes are updated as necessary, the accessor is closed, and the activity's state is changed to READY. The activity is then offered to sessions as permitted by the delegation assignment rule.

NOTE The *state* parameter is used to confirm that the current state of the activity is what you expect when you issue the `DelegateActivity` command. If the state has changed, the engine does not execute the command.

The *user_name* and *other_info* arguments must be supplied if required by the delegation assignment rule. For example, if a delegation rule named "ManagerOf" looks for the manager of the delegator, then the delegator's user name and other info (in this case the manager's name) would be passed to the `ManagerOf` assignment rule. This behavior is similar in functionality to the linked activity mechanism described in the *iIS Process Development Guide*.

Once the delegated user has completed (or aborted) the activity, process execution continues as specified in the process definition.

DeleteFilter

The `DeleteFilter` command deletes the specified event filter in the current engine.

```
DeleteFilter filter_id
```

Argument	Description
<i>filter_id</i>	The specified filter.

The `DeleteFilter` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

You can to use the `ListFilters` Script command to display a list of filters before selecting the filter to delete.

FindDBService

The `FindDBService` command sets the specified database service for the current engine as the current component.

```
FindDBService service_name
```

Argument	Description
<i>service_name</i>	The specified database service.

The `FindDBService` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine. The specified database service must also be running.

The `FindDBService` command is normally used to check the status of a specific database service (using the `ShowStatus` Script command) or to shut it down (using the `Shutdown` Script command).

FindEngine

The `FindEngine` command sets the specified engine as the current engine.

```
FindEngine engine_name
```

Argument	Description
<code>engine_name</code>	The specified engine.

The `FindEngine` command is always available. The specified engine need not be running.

You can use the `ListEngines` Script command to display the list of engines. The `FindEngine` command is normally the first command issued to perform any engine management, database management, or process execution management tasks.

FindGovernor

The `FindGovernor` command sets the governor for the current engine as the current component.

```
FindGovernor
```

The `FindGovernor` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine. The governor must also be running.

The `FindGovernor` command is normally used to check the status of the governor (using the `ShowStatus` Script command) or to shut it down (using the `Shutdown` Script command).

FindNode

The `FindNode` command sets the specified node as the current node.

```
FindNode node_name
```

Argument	Description
<i>node_name</i>	The specified node. Must be in same environment as the engine.

The `FindNode` command is available only for the current engine.

The `FindNode` command is normally used to start engine components (using the `StartGovernor`, `StartDBService`, and `StartUnit` Script commands).

FindParentEngine

The `FindParentEngine` command makes the component's parent engine current.

```
FindParentEngine
```

The `FindParentEngine` command is available only when an engine component is current.

The `FindParentEngine` command is used to navigate to the current engine when one of its components is current and you want to make another of its components current. (The `Find` and `Start` component Script commands are only available when an engine is current.)

FindPrimary

The `FindPrimary` command sets the primary engine unit for the current engine as the current component.

```
FindPrimary
```

The `FindPrimary` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine. The primary engine unit must also be running.

The `FindPrimary` command is normally used to check the status of the primary engine unit (using the `ShowStatus` Script command), to change its state (using the `SetState` and `Shutdown` Script commands), and to perform process execution and engine database management tasks.

FindUnit

The `FindUnit` command sets the specified engine unit for the current engine as the current component.

```
FindUnit unit_name
```

Argument	Description
<i>unit_name</i>	The specified engine unit.

The `FindUnit` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine. The specified engine unit must also be running.

The `FindUnit` command is normally used to check the status of a specific engine unit (using the `ShowStatus` Script command) or to change its state (using the `SetState` and `Shutdown` Script commands). If the engine unit is the primary engine unit then you can also perform process execution and engine database management tasks.

FlushLog

The `FlushLog` command removes all the data held in the history log database tables of the current engine.

The `FlushLog` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

You can use the `FlushLog` command when the history log database grows too large. Typically you would first back up the current database or export the history log tables.

IIOPServer

The `IIOPServer` command starts, stops, or displays information about the IIOP service needed to support CORBA/IIOP iIS process client applications.

```
IIOPServer [start|stop|show]
```

The `IIOPServer` command is always available.

The `IIOPServer` command is used to start up the IIOP service installed by the iIS installer program on the central server node in your iPlanet UDS environment. The IIOP service is required for CORBA/IIOP client applications to access an iIS process engine.

ListActivities

The `ListActivities` command displays all activities in the current engine.

```
ListActivities
```

The `ListActivities` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `ListActivities` command displays activities in all process instances of all registered process definitions. For each activity, the `ListActivities` command displays the following information: activity name, activity ID, activity state, and process name.

ListActivityQueues

The `ListActivityQueues` command displays queues either for the specified process or for all processes in the current engine.

```
ListActivityQueues [process_name]
```

Argument	Description
<i>process_name</i>	The specified process. If not specified then all queues in the engine are displayed.

The `ListActivityQueues` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `ListActivityQueues` command displays all activity queues in a specified process definition. For each queue the `ListActivityQueues` command displays the process name and the queue name.

ListConductorDistributions

The `ListConductorDistributions` command displays a list of the specified type of library distribution (generated from iIS process development workshops) available for registration.

```
ListConductorDistributions [type]
```

Argument	Description
<i>type</i>	The specified distribution type: process definition, assignment rule dictionary, or user profile.

The `ListConductorDistributions` command is always available. If the distribution type is not specified, all types are listed, grouped by type.

The `ListConductorDistributions` command is used to display a list of library distributions of the appropriate type that you might want to register using the `Register` Script command. For each distribution, the `ListConductorDistributions` command displays the distribution name and the distribution type.

You can generate the libraries listed by this command using the `FindPlan` FScript command to make a specified plan current, and then issuing the `MakeConductorDistribution` Script command to create the library distribution on the central server of the active environment. The list of iIS libraries accessed by `ListConductorDistributions` is maintained by the `WFEngAgent` service running on the central server node.

ListEngines

The `ListEngines` command displays a list of all running and configured engines in the current environment.

`ListEngines`

The `ListEngines` command is always available.

The list displayed includes all configured engines, indicating engines that are ONLINE. It also includes engines that are ONLINE for which there is no configuration file.

ListFilters

The `ListFilters` command displays all sieves in the current engine.

`ListFilters`

The `ListFilters` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

For each filter the `ListFilters` command displays the filter ID, the time interval, and the filter expression.

ListProcesses

The `ListProcesses` command displays process instances for a specified process (or displays all process instances) in the current engine, or lists a summary of process instances by process name.

`ListProcesses` [short] [name *process_name*]

Argument	Description
<i>process_name</i>	The specified process.

The `ListProcesses` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `ListProcesses` command displays process instances.

Options

- If you use the `short` option, the command displays a summary of process instances by process definition.
- If you use the `name` option, the command displays all process instances for the specified registered process definition.
- If you do not specify an option, it displays all process instances in the current engine.

For any given process instance, you can monitor or manage its activities, timers, attributes, and attribute locks. For each process instance, the `ListProcesses` command displays the following information: process name, the process definition version, the value of the primary process attribute, the primary process attribute name, the process instance ID, and the process creation time.

ListRegistrations

The `ListRegistrations` command displays distributions of the specified type (or all types) registered with the current engine.

```
ListRegistrations [type]
```

Argument	Description
<code>type</code>	The specified component type: process definition, assignment rule dictionary, user profile, uservalidation, or alias.

The `ListRegistrations` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine. If the type is not specified, all types are listed, grouped by type.

You can delete any registered entity from the current engine's registration database using the appropriate `UnRegister` Script command. You can register additional entities using the appropriate `Register` Script command. For each registered entity the `ListRegistrations` command displays the name of the registered component and the type.

ListSessions

The `ListSessions` command displays all sessions maintained by the current engine.

`ListSessions`

The `ListSessions` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

For any given session, you can change its state using the `SuspendSession` or `TerminateSession` Script command. For each session, the `ListSessions` command displays the following information: session name, session ID, user, session state (ACTIVE or SUSPENDED), and the number of items on the session activity list.

ListTimers

The `ListTimers` command displays all timers in the current engine.

`ListTimers`

The `ListTimers` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `ListTimers` command displays timers in all process instances of all registered process definitions. For each timer, the `ListTimers` command displays the timer name, the process ID, the state, and the expiration time and date.

ListTransactions

The `ListTransactions` command lists all in-progress iIS engine transactions for all sessions for a specified engine. For details on two-phase commit operations, see [“Monitoring and Managing Two-Phase Commit” on page 199](#).

To roll back an in-progress iIS engine transaction, use the `RollbackTransaction` command. To commit an in-progress iIS engine transaction, use the `CommitTransaction` command.

MakeConductorDistribution

The `MakeConductorDistribution` command generates a library distribution for the current plan. The plan must be created in the iIS process development workshops.

`MakeConductorDistribution`

The `MakeConductorDistribution` command is always available; however, it requires that a plan created in the iIS process development workshops also be current. To make such a plan current, use the `FindPlan` Script command (see [“Making iIS Library Distributions” on page 243](#)).

NOTE Unlike the generation of library distributions in the iPlanet UDS development environment, generating library distributions with `MakeConductorDistribution` automatically increases the compatibility level of the current plan. iIS developers must keep track of the compatibility level of any iIS plans they export, noting that their compatibility level will change after generation of library distributions.

After issuing the `MakeConductorDistribution` command, you can use the `ListConductorDistributions` Script command to confirm that the library distribution has been generated.

ModLogFlags

The `ModLogFlags` command modifies the log message filters set for the current component.

`ModLogFlags {+(message_filters) | -(message_filters)}`

Argument	Description
<code>+(message_filters)</code>	Turn on the log flag settings given in the parentheses.
<code>-(message_filters)</code>	Turn off the log flag settings given in the parentheses.

The `ModLogFlags` command is only available when an engine component is current.

To start logging, use the '+' followed by a set of message filters in parentheses. To stop logging, use the '-' followed by a set of message filters in parentheses.

The settings specified with the `ModLogFlags` command modify the log message filters that were set when the current component started up, based on the `FORTE_LOGGER_SETUP` environment variable.

An example of the `ModLogFlags` command is:

```
cscript> ModLogFlags +(trc:wr:2:2)
```

This filter specifies that all messages about sessions created in the engine—but not a lot of detail—be written to the primary engine unit's log file (on the server node on which the primary engine unit is running). For information on iIS message filters, see [“Messages and Message Filters” on page 212](#).

ReadyActivity

The `ReadyActivity` command changes the state of the specified activity from PENDING to READY.

```
ReadyActivity process_id activity_name
```

Argument	Description
<i>process_id</i>	The process instance in which the activity was created.
<i>activity_name</i>	The name of the activity.

The `ReadyActivity` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `ReadyActivity` command does not apply to activities that do not pass through READY state, such as subprocess activities.

Before changing the state of an activity from PENDING to READY, the engine first evaluates the activity's Ready expression if one is specified in the process definition.

RegisterAlias

The `RegisterAlias` command registers the specified subprocess alias in the current engine's registration database. For more information on aliases, see ["About Aliases" on page 143](#).

```
RegisterAlias alias_name process_name engine_name environment
```

Argument	Description
<i>alias_name</i>	The name of the alias (same as the process name).
<i>process_name</i>	The name of the process referenced by the specified alias.
<i>engine_name</i>	The name of the engine on which the specified process is executed.
<i>environment</i>	The name of the environment in which the specified engine is located.

The `RegisterAlias` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `RegisterAlias` command registers an alias only if an alias of the same name is not already registered.

RegisterAssignmentRules

The `RegisterAssignmentRules` command registers the specified assignment rule dictionary (or all available assignment rule dictionaries) in the current engine's registration database.

```
RegisterAssignmentRules [dictionary_name]
```

Argument	Description
<i>dictionary_name</i>	The name of the assignment rule dictionary to be registered. If not specified, then all available assignment rule dictionaries are registered.

The `RegisterAssignmentRules` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

Use the `ListConductorDistributions` Script command to display a list of distributions available for registration (you can filter the list for assignment rule dictionaries). The `RegisterAssignmentRules` command registers an assignment rule dictionary only if a dictionary of the same name and compatibility level is not already registered.

RegisterProcessDefinition

The `RegisterProcessDefinition` command registers the specified process definition (or all available process definitions) in the current engine's registration database.

```
RegisterProcessDefinition [process_name]
```

Argument	Description
<i>process_name</i>	The name of the process definition to be registered. If not specified, then all available process definitions are registered.

The `RegisterProcessDefinition` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

Use the `ListConductorDistributions` Script command to display a list of distributions available for registration. You can filter the list for process definitions. The `RegisterProcessDefinition` command registers a process definition only if a process definition of the same name and compatibility level is not already registered.

RegisterUserProfile

The `RegisterUserProfile` command registers a user profile in the current engine's registration database.

```
RegisterUserProfile [user_profile_name]
```

Argument	Description
<i>user_profile_name</i>	The name of the user profile to be registered. If not specified, then all available user profiles are registered.

The `RegisterUserProfile` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

Use the `ListConductorDistributions` Script command to display a list of distributions available for registration. You can filter the list for user profiles. You can register multiple user profiles with an engine if each has a unique name, but you can register only one compatibility level of a user profile of a given name. The `RegisterUserProfile` command registers a user profile only if a user profile of the same name is not already registered.

RegisterValidation

The `RegisterValidation` command registers a validation in the current engine's registration database.

```
RegisterValidation validation_name
```

Argument	Description
<code>validation_name</code>	The name of the validation.

The `RegisterValidation` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

Use the `ListConductorDistributions` Script command to display a list of distributions available for registration. You can filter the list for validations. You can only register one validation with an engine (any previous registration will be implicitly unregistered). The `RegisterValidation` command registers a validation only if a validation of the same name and compatibility level is not already registered.

RemoveReadLock

The `RemoveReadLock` command removes a read lock on the specified process attribute.

```
RemoveReadLock process_id attribute_name
```

Argument	Description
<i>process_id</i>	The process instance of the specified attribute.
<i>attribute_name</i>	The name of the process attribute that is locked.

The `RemoveReadLock` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `RemoveReadLock` command lets you remove a read lock that remains due to some unusual circumstance. Completing or aborting an activity should remove any locks placed on a process attribute; however, if a session is unexpectedly terminated, locks might remain in place.

RemoveWriteLock

The `RemoveWriteLock` command removes a write lock on the specified process attribute.

```
RemoveWriteLock process_id attribute_name
```

Argument	Description
<i>process_id</i>	The process instance of the specified attribute.
<i>attribute_name</i>	The name of the process attribute that is locked.

The `RemoveWriteLock` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `RemoveWriteLock` command lets you remove a write lock that remains due to some unusual circumstance. Completing or aborting an activity should remove any locks placed on a process attribute; however, if a session is unexpectedly terminated, locks might remain in place.

RollbackActivity

The `RollbackActivity` command changes the state of the specified activity from `ACTIVE` to `READY`.

```
RollbackActivity process_id activity_name
```

Argument	Description
<code>process_id</code>	The process instance in which the activity was created.
<code>activity_name</code>	The name of the activity.

The `RollbackActivity` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `RollbackActivity` command does not apply to activities that do not pass through `READY` state, such as `SubProcess`, `First`, and `Last` activities.

When you issue the `RollbackActivity` command, the engine performs the following actions:

- Discards pending process attribute updates—rolls back changes that were made in performing the activity and frees locks on process attributes
- Changes the activity state from `ACTIVE` to `READY`
- Posts an `ActivityListUpdate` event to all sessions associated with the activity
- If the activity is an offered activity, the engine reassigns it to sessions based on the activity's assignment rules
- If the activity is a queued activity, the engine places it back on the queue to which it was assigned
- If the activity is an automatic activity, the engine automatically places it in `ACTIVE` state, and re-evaluates the `AutoAction` expression for the activity.

RollbackTransaction

The `RollbackTransaction` command rolls back (undoes) an in-progress iIS engine transaction (that is, one in a `PREPARE` phase). For details on two-phase commit operations, see [“Monitoring and Managing Two-Phase Commit” on page 199](#).

```
RollbackTransaction {session_ID | transaction_ID}
```

Argument	Description
<i>session_ID</i>	The identifier of the session which has a transaction in a PREPARE phase. A session can have only one in-progress two-phase transaction.
<i>transaction_ID</i>	The identifier of the transaction in a PREPARE phase.

To commit an in-progress iIS engine transaction, use the `CommitTransaction` command. To list all in-progress iIS engine transactions for a given engine, use the `ListTransactions` command.

SendMessage

The `SendMessage` command sends a message to a specified sessions on the current engine.

```
SendMessage session_id message urgency
```

Argument	Description
<i>session_id</i>	The specified session.
<i>message</i>	An alphanumeric string that constitutes the text of the message. Message text must be enclosed in double quotes.
<i>urgency</i>	One of two specified string constants: INFORMATIONAL or CRITICAL.

The `SendMessage` command is only available when an engine is current. Use the `FindEngine` Script command to set the current engine.

The `SendMessage` command is used to notify a session about a pending administrative action, such as a request for consultation, pending suspension of the session, and so forth.

SetAttributeValue

The `SetAttributeValue` command sets a specified process attribute in the current engine to a specified value.

```
SetAttributeValue process_id attribute_name attribute_type value
```

Argument	Description
<i>process_id</i>	The process instance of the specified attribute.
<i>attribute_name</i>	The name of the process attribute that is to be set.
<i>attribute_type</i>	The data type of the attribute value.
<i>value</i>	The value to be assigned to the attribute. String values must be enclosed in within double quotes.

The `SetAttributeValue` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `SetAttributeValue` command lets you change a specified process attribute value of a process instance (for example, a priority attribute, which affects the routing between activities).

SetPassword

The `SetPassword` command sets password protection for opening Conductor Script in your active environment. After you set a password, other users cannot open Conductor Script (or iIS Console, or iPlanet UDS Environment Console, or iPlanet UDS Escript) in that environment without using the password.

```
SetPassword [old_password] [new_password]
```

Argument	Description
<i>old_password</i>	The current existing password.
<i>new_password</i>	The new (or null) password.

The `SetPassword` command is always available. The specified engine need not be running to use the `SetPassword` command.

You can use the `SetPassword` Script command to set a password, change an existing password to a new password, or delete an existing password.

SetPrimary

The `SetPrimary` command sets the specified engine unit for the current engine as the primary unit.

```
SetPrimary unit_name
```

Argument	Description
<code>unit_name</code>	The specified engine unit.

The `SetPrimary` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine. The specified engine unit must also be running.

The `SetPrimary` command is normally used to switch the role of the primary and backup engine units. If the specified engine unit is the backup unit, the primary unit will be placed in a STANDBY state and the backup unit will then be placed ONLINE. If the specified engine unit is the primary unit, nothing will be done.

The `SetPrimary` command makes the specified engine unit the current component.

SetQueuedActivityPriority

The `SetQueuedActivityPriority` command reprioritizes activities in an activity queue by setting the prioritizing process attribute for a specified activity to a specified value.

```
SetQueuedActivityPriority process_id activity_id prioritizing_value
```

Argument	Description
<code>process_id</code>	The process instance in which the activity was created.
<code>activity_id</code>	The activity ID of the specified activity.
<code>prioritizing_value</code>	The value to be assigned to the queue prioritizing process attribute of the process instance.

The `SetQueuedActivityPriority` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `SetQueuedActivityPriority` command lets you change the ordering of activities in a queue by changing the value of the queue prioritizing attribute for a specified activity in the queue. Doing so causes the engine to place the specified activity instance in a different position in the queue.

SetState

The `SetState` command sets the state of the specified engine unit.

```
SetState state
```

Argument	Description
<i>state</i>	The state to which the engine unit will be set: ONLINE or STANDBY.

The `SetState` command is only available if the current component is an engine unit. Use the `FindEngine` Script command to set the current engine and the `FindUnit` Script command to make the desired engine unit current.

The `SetState` command is normally used to take the primary engine unit offline for administrative functions and then to restore it to ONLINE state afterwards. Placing the primary engine unit in STANDBY state will not cause the partner engine unit to come ONLINE.

SetTimer

The `SetTimer` command sets the specified timer to the specified state and resumes operation (or resets the timer).

```
SetTimer process_id timer_name state [reset]
```

Argument	Description
<i>process_id</i>	The process instance of the specified timer.
<i>timer_name</i>	The name of the specified timer.
<i>state</i>	The specified new state: ON, OFF, or no change.

The `SetTimer` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

You can use the `SetTimer` command to turn any timer on or off, to reset an elapsed timer to its original time interval, or to do both. To set new timer expiration times, use the `SetTimerDeadline` or `SetTimerElapsed`, depending on the type of timer. The `reset` option lets you use the `SetTimer` command to reset the timer, rather than resume operation. This option applies only to elapsed timers.

SetTimerDeadline

The `SetTimerDeadline` command sets the specified deadline timer to the specified state (ON, OFF, or no change) and sets a new expiration time.

```
SetTimerDeadline process_id timer_name state expiration_time
```

Argument	Description
<i>process_id</i>	The process instance of the specified timer.
<i>timer_name</i>	The name of the specified timer.
<i>state</i>	The specified new state: ON, OFF, or no change.
<i>expiration_time</i>	The new expiration time in DateTimeData format (dd-nnn-yyyy hh:mm:ss, for example, 03-Feb-1993 22:45:12). Applies only to deadline timers.

The `SetTimerDeadline` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

You can use the `SetTimerDeadline` command to turn any deadline timer on or off, and to set a new expiration time.

SetTimerElapsed

The `SetTimerElapsed` command sets the specified elapsed timer to the specified state (ON, OFF, or no change) and sets a new elapsed time interval until expiration.

```
SetTimerElapsed process_id timer_name state time_interval
```

Argument	Description
<i>process_id</i>	The process instance of the specified timer.
<i>timer_name</i>	The name of the specified timer.
<i>state</i>	The specified new state: ON, OFF, or no change.
<i>time_interval</i>	The new time interval until expiration, in IntervalData format (years:months:days:hours:minutes:seconds:milliseconds). Applies only to elapsed timers.

The `SetTimerElapsed` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

You can use the `SetTimerElapsed` command to turn any elapsed timer on or off, and to set a new elapsed time interval until expiration.

ShowActivity

The `ShowActivity` command displays the properties of the specified activity in the current engine.

```
ShowActivity process_id activity_name
```

Argument	Description
<i>process_id</i>	The process instance in which the activity was created.
<i>activity_name</i>	The name of the activity.

The `ShowActivity` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `ShowActivity` command displays the following information:

- activity name
- activity ID
- process name
- process ID
- activity state
- activity type
- application description
- list of sessions for which activity is on session activity list, displaying session ID
- list of process attributes, displaying: attribute name, attribute type, attribute value, attribute lock status
- list of timer links, displaying: timer link name, activity state, timer state, value

You can use the `ListActivities` Script command to display a list of activities in the engine before selecting the activity to display.

ShowActivityQueue

The `ShowActivityQueue` command displays the contents either of the specified queue or of all queues for a given process definition executed by the current engine.

```
ShowActivityQueue process_name [queue_name]
```

Argument	Description
<i>process_name</i>	The name of the process definition in which the queued activity is defined.
<i>queue_name</i>	The name of the specified activity queue (the same as the name of the queued activity). If not specified, then the contents of all queues for the specified process are displayed.

The `ShowActivityQueue` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `ShowActivityQueue` command displays a list of activities in the queue, if any, or of sessions waiting for a queued activity if no activities are in the queue. For each activity in the queue, the `ShowActivityQueue` command displays the following information: activity name, activity ID, activity state, prioritizing value (value of the queue prioritizing process attribute), and process ID.

ShowConfiguration

The `ShowConfiguration` command displays contents of the configuration file of the current engine.

```
ShowConfiguration
```

The `ShowConfiguration` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine. No engine components need be running for this command to work.

ShowEngine

The `ShowEngine` command displays the configuration, state, and other properties of the specified engine.

```
ShowEngine engine_name
```

Argument	Description
<i>engine_name</i>	The specified engine.

The `ShowEngine` command is always available. The specified engine need not be running to use the `ShowEngine` command.

The `ShowEngine` command displays the following properties:

- engine name
- governor run state: TRUE or FALSE
- primary engine unit run state: TRUE or FALSE
- number and names of engine units running
- number and names of database services running

You can use the `ListEngines` Script command to display a list of engines in the environment before selecting the engine to display.

ShowLogFlags

The `ShowLogFlags` command displays the log message filters (log flags) for the current component.

```
ShowLogFlags
```

The `ShowLogFlags` command is available only when an engine component is current.

For information on IIS message filters, see [“Messages and Message Filters” on page 212](#).

ShowProcess

The `ShowProcess` command displays the properties of the specified process instance in the current engine.

```
ShowProcess process_id
```

Argument	Description
<i>process_id</i>	The specified process instance.

The `ShowProcess` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `ShowProcess` command displays the following information:

- process name
- process compatibility level
- process ID
- creation time
- list of process attributes, displaying: attribute name, attribute type, attribute value, attribute lock status

- list of activities, displaying activity name, activity ID, process ID, activity state (as in `ListActivities` Script command)
- list of timers, displaying timer name, process ID, state, type, expiration time and date (as in `ListTimers` Script command)
- parent activity and engine (for subprocesses)
- list of active attribute accessors

You can use the `ListProcesses` Script command to display a list of process instances in the engine before selecting the process instance to display.

ShowSession

The `ShowSession` command displays the properties of the specified session in the current engine.

```
ShowSession session_id
```

Argument	Description
<i>session_id</i>	The specified session.

The `ShowSession` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `ShowSession` command displays the following information:

- session ID
- session name
- client connected: TRUE or FALSE
- session state: ACTIVE or SUSPENDED
- version
- administrative user: TRUE or FALSE
- activity list size

- user
- user's roles
- list of active attribute accessors

You can use the `ListSessions` Script command to display a list of sessions in the engine before selecting the session to display.

ShowStatus

The `ShowStatus` command displays status information for the current engine or any of its individual components.

`ShowStatus [short]`

The `ShowStatus` command is available only when the engine is current or if an engine component is current. Use the `FindEngine` Script command to set the current engine, and, in addition, the `FindGovernor`, `FindDBService`, `FindUnit`, or `FindPrimary` Script command to make the corresponding engine component current.

The properties displayed by the `ShowStatus` command depend upon whether the engine is current or an engine component is current. If an engine component is current, then using the `short` option of the `ShowStatus` command will display an abbreviated set of properties for that component (italicized properties in the lists below are not displayed)

engine:

- engine name
- governor run state: TRUE or FALSE
- primary engine unit run state: TRUE or FALSE
- number and names of engine units running
- number and names of database services running
- number of idle database services

engine unit:

(See also “**primary engine unit:**” on page 311.)

- unit name
- unit state: ONLINE or STANDBY
- shell state: IDLE, TOKEN_REQUESTED (negotiating with partner), or CONNECTED
- governor channel: OPEN or CLOSED
- partner channel: OPEN or CLOSED
- ping interval
- *start time*
- *host node*

database service:

- database service name
- database connection state: CONNECTED or DISCONNECTED
- router channel: OPEN or CLOSED
- ping interval
- *start time*
- *host node*
- *work units processed*

governor:

- governor state: E1, BOTH, E2, or IDLE
- ping interval
- *start time*
- *host node*

primary engine unit:

- unit name
- startup phase (if not ONLINE)
- unit state: ONLINE or STANDBY
- shell state: IDLE, TOKEN_REQUESTED (negotiating with partner), or CONNECTED
- governor channel: OPEN or CLOSED
- partner channel: OPEN or CLOSED
- ping interval
- *start time*
- *host node*
- *instruments*

The following table shows the instrumentation displayed for the primary engine unit.

Instrument	Description
ActiveSessions	Number of current active sessions
SuspendedSessions	Number of current suspended sessions
CurrentSessions	Number of current sessions = ActiveSessions +SuspendedSessions
TotalSessions	Number of sessions since engine startup = CurrentSessions + Number of terminated sessions
ActiveProcessInstances	Number of current active process instances
TotalProcessInstances	Number of process instances since engine startup = ActiveProcess Instances + number of terminated process instances
ActiveSubProcessInstances	Number of current active subprocess instances
TotalSubProcessInstances	Number of subprocess instances since engine startup = ActiveSubProcess Instances + number of terminated subprocess instances
PendingActivities	Number of current PENDING activities
ReadyActivities	Number of current READY activities

Instrument	Description
ActiveActivities	Number of current ACTIVE activities
CurrentActivities	Number of current activities = PendingActivities + ReadyActivities + ActiveActivities
CompletedActivities	Number of completed activities (terminated) since engine startup
AbortedActivities	Number of aborted activities (terminated) since engine startup
TotalActivities	Number of activities since engine startup = CurrentActivities + CompletedActivities + AbortedActivities
RunningTimers	Number of current running timers
StoppedTimers	Number of current stopped timers
CurrentTimers	Number of current timers = RunningTimers + StoppedTimers
TotalTimers	Number of timers since engine startup = CurrentTimers + number of terminated timers
ReadLocks	Number of current read (shared) locks
WriteLocks	Number of current write (exclusive) locks
CurrentLocks	Number of current locks = ReadLocks + WriteLocks
TotalLocks	Number of locks since engine startup = CurrentLocks + number of terminated locks
QueuedReadLocks	Number of current read (shared) locks queued
QueuedWriteLocks	Number of current write (exclusive) locks queued
ResponseTime	The average time it takes for the engine to complete a request made by a session
ActivePages	The size of the primary engine unit memory heap
AllocatedPages	The amount of the active memory currently allocated to objects created by the primary engine unit
PeakAllocatedPages	The amount of allocated memory remaining after the most recent memory reclamation (probably the best measure of active memory utilization)
TransactionCount	The number of transactions completed with the engine database since engine startup.
DatabaseName	The name of the engine database.

Instrument	Description
LoginUserName	The database user (login) name.
LoggingState	The database logging settings: the tables to which the engine is logging state and history information.
TransactionCommitTime	The average time it takes the engine to commit a change of state (complete a transaction involving the engine database).
DatabaseType	The database management system: ORACLE, SYBASE, etc.
DBServiceRequestTime	The average time it takes for the database service to complete a request made by the engine.

ShowTimer

The `ShowTimer` command displays the properties of the specified timer.

```
ShowTimer process_id timer_name
```

Argument	Description
<i>process_id</i>	The process instance of the specified timer.
<i>timer_name</i>	The specified timer.

The `ShowTimer` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `ShowTimer` command displays the following information:

- timer name
- timer state
- process name and ID
- type
- started: TRUE or FALSE
- expiration time and date

You can use the `ShowProcess` Script command to display a list of timers in a specified process instance before selecting the timer to display.

Shutdown

The `Shutdown` command shuts down the current engine or one of its individual components.

`Shutdown`

The `Shutdown` command is available only when the engine is current or if an engine component is current. Use the `FindEngine` Script command to set the current engine, and, in addition, the `FindGovernor`, `FindDBService`, `FindUnit`, or `FindPrimary` Script command to make the corresponding engine component current.

The effect of the `Shutdown` command depends upon whether the engine is current or an engine component is current:

- if engine: shuts down all components of current engine
- if engine component: shuts down the current component

If the `Shutdown` command results in a primary engine unit being shut down, then all state information in the engine is lost and must be recovered from the engine database when a primary unit comes online.

StartActivity

The `StartActivity` command changes the state of the specified activity from `READY` to `ACTIVE`.

`StartActivity process_id activity_name session_id`

Argument	Description
<code>process_id</code>	The process instance in which the activity was created.
<code>activity_name</code>	The name of the activity.
<code>session_id</code>	The session that will perform the <code>ACTIVE</code> activity. Applies only to <code>Offered</code> and <code>Queued</code> activities.

The `StartActivity` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `StartActivity` command does not apply to activities that do not pass through `ACTIVE` or `READY` state, such as `First` and `Last` activity.

In the case of offered and queued activities, which are performed by client applications, the engine will assign the ACTIVE activity to the session specified by the `session_id` parameter. In the case of Automatic or SubProcess activities, which are not directly performed by client applications, the engine need not assign the ACTIVE activity to a session.

StartDBService

The `StartDBService` command starts the specified database service for the current engine.

```
StartDBService service_name priority
```

Argument	Description
<code>service_name</code>	The specified database service.
<code>priority</code>	An integer (positive or negative) that assigns a priority to the specified service. A higher numeric value signifies a higher priority. A default priority of "1" is used if no priority is specified. See "Full Configuration: Failover and Load Balancing Combined" on page 35.

The `StartDBService` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `StartDBService` command reads the memory flag setting for database service specified in the engine configuration file (see ["Engine Configuration File"](#) on page 90 and [Step 10](#) under ["How to Configure a New Engine"](#) on page 95).

The `ShowStatus` Script command—or the `FindDBService` command followed by the `ShowStatus` command—can be used to check that the database service has started up properly.

StartEngine

The `StartEngine` command starts all engine components as specified in the engine configuration file (see ["Engine Configuration File"](#) on page 90).

```
StartEngine [newLog] [newState] [newRegistration] [cold]
```

The `StartEngine` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `StartEngine` command includes a number of start options that create new engine database tables:

Table A-1 StartEngine Command Options

Option	Description
<code>newLog</code>	New history log database tables are created during the primary engine unit startup process. All history log information stored in an existing database is lost.
<code>newState</code>	New current state database tables are created during the primary engine unit startup process. All current state information stored in an existing database is lost.
<code>newRegistration</code>	New registration database tables are created during the primary engine unit startup process. All registration information stored in an existing database is lost. You must also specify <code>newState</code> when you specify the <code>newRegistration</code> option.
<code>cold</code>	All new database tables (log, state, and registration) are created during the primary engine unit startup process. The <code>cold</code> option must always be used the first time an engine is started. If used in subsequent engine starts, however, the <code>cold</code> option will cause deletion of the existing engine database before the creation of the new one. All current state information, history log data, and registration information stored in an existing database is lost.

The `newLog`, `newState`, and `newRegistration` options are independent, except that you cannot create new registration tables without also creating new state tables.

The `ShowStatus` Script command can be used to check that the engine has started up properly.

StartGovernor

The `StartGovernor` command starts the governor for the current engine.

```
StartGovernor
```

The `StartGovernor` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `StartGovernor` command reads the memory flag setting for governor specified in the engine configuration file (see “[Engine Configuration File](#)” on page 90 and [Step 10](#) under “[How to Configure a New Engine](#)” on page 95).

The `ShowStatus` Script command—or the `FindGovernor` command followed by the `ShowStatus` command—can be used to check that the governor has started up properly.

StartTimer

The `StartTimer` command changes the state of a specified timer from OFF to ON.

```
StartTimer process_id timer_name
```

Argument	Description
<i>process_id</i>	The process instance of the specified timer.
<i>timer_name</i>	The name of the timer to be started.

The `StartTimer` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `StartTimer` command lets you start the specified timer or restart it if it has previously been turned off.

StartUnit

The `StartUnit` command starts the specified engine unit for the current engine.

```
StartUnit unit_name [newLog] [newState] [newRegistration] [cold]
```

Argument	Description
<i>unit_name</i>	The specified engine unit.

The `StartUnit` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `startUnit` command includes the following start options that create new engine database tables:

Table A-2 StartUnit Command Options

Option	Description
<code>newLog</code>	New history log database tables are created during the primary engine unit startup process. All history log information stored in an existing database is lost.
<code>newState</code>	New current state database tables are created during the primary engine unit startup process. All current state information stored in an existing database is lost.
<code>newRegistration</code>	New registration database tables are created during the primary engine unit startup process. All registration information stored in an existing database is lost. You must also specify <code>newState</code> when you specify the <code>newRegistration</code> option.
<code>cold</code>	All new database tables (log, state, and registration) are created during the primary engine unit startup process. The <code>cold</code> option must always be used the first time an engine is started. If used in subsequent engine starts, however, the <code>cold</code> option causes deletion of the existing engine database before the creation of the new one. All current state information, history log data, and registration information stored in an existing database is lost.

The `newLog`, `newState`, and `newRegistration` options are independent, except that you cannot create new registration tables without also creating new state tables.

The `startUnit` command reads the memory flag setting for the engine unit specified in the engine configuration file (see [“Engine Configuration File” on page 90](#) and [Step 10](#) under [“How to Configure a New Engine” on page 95](#)).

The `ShowStatus` Script command— or the `FindUnit` command followed by the `ShowStatus` command—can be used to check that the engine unit has started up properly.

StopTimer

The `StopTimer` command changes the state of the specified timer from ON to OFF.

```
StopTimer process_id timer_name
```

Argument	Description
<code>process_id</code>	The process instance of the specified timer.
<code>timer_name</code>	The name of the timer to be reset.

The `StopTimer` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `StopTimer` command lets you stop the specified timer. The timer can be subsequently restarted with the `StartTimer` Script command.

SuspendAllSessions

The `SuspendAllSessions` command suspends all active sessions in the current engine.

```
SuspendAllSessions
```

The `SuspendAllSessions` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `SuspendAllSessions` command places all `ACTIVE` sessions in `SUSPENDED` state, making them dormant: the engine cannot assign new activities to suspended sessions or post events to them.

Automatic reconnect by the client application does not take place if sessions are suspended with the `SuspendSession` command. A subsequent explicit client request to open each session can restore the session to `ACTIVE` state or create a new session, depending on the value of the `reconnectAction` property set when each session was originally opened.

The engine posts a `SessionSuspended` event on each session before it is suspended, so that the client applications can take appropriate action. The effect of suspending a session on any `ACTIVE` activities depends on the `suspendAction` property set by each client application when accepting each activity.

SuspendSession

The `SuspendSession` command suspends the specified active session, or sessions, in the current engine.

```
SuspendSession session_id [session_id...]
```

Argument	Description
<code>session_id</code>	The specified active session. (A list of up to ten <code>session_ids</code> separated by spaces is supported.)

The `SuspendSession` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `SuspendSession` command places an `ACTIVE` session in `SUSPENDED` state. This means that the session is now dormant: the engine cannot assign new activities to a suspended session or post events to it.

Automatic reconnect by the client application does not take place if a session is suspended using the `SuspendSession` command. A subsequent explicit client request to open the session can restore it to `ACTIVE` state or create a new session, depending on the value of the `reconnectAction` property set when the session was first opened.

The engine posts a `SessionSuspended` event on the session before it is suspended, so that the client application can take appropriate action. The effect of suspending a session on any `ACTIVE` activities depends on the `suspendAction` property set by the client application when accepting each activity.

TerminateAllSessions

The `TerminateAllSessions` command terminates all sessions in the current engine.

```
TerminateAllSessions
```

The `TerminateAllSessions` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `TerminateAllSessions` command ends all sessions and deletes them from the engine and the engine's state database table.

The engine posts a `SessionTerminated` event on each session before it is terminated, so that the respective client applications can take appropriate action. The effect of terminating a session on any `ACTIVE` activities is to place the activities back in `READY` state.

TerminateSession

The `TerminateSession` command terminates the specified session in the current engine.

```
TerminateSession session_id [session_id...]
```

Argument	Description
<i>session_id</i>	The specified session. (A list of up to ten <i>session_ids</i> separated by spaces is supported.)

The `TerminateSession` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

The `TerminateSession` command ends the session and deletes it from the engine and the engine's state database table.

The engine posts a `SessionTerminated` event on the session before it is terminated, so that the client application can take appropriate action. The effect of terminating a session on any `ACTIVE` activities is to place the activities back in a `READY` state.

Uninstall

The `Uninstall` command deletes an iIS distribution of the specified name and compatibility level from the environment repository.

```
Uninstall name compatibility_level
```

Argument	Description
<i>name</i>	The name of the library distribution to be deleted from the environment repository.
<i>compatibility_level</i>	The compatibility level of the distribution to be deleted.

The `Uninstall` command is always available.

The `Uninstall` command is normally performed to clean up the environment repository after unregistering an iIS distribution from all engines in an environment.

UnRegisterAlias

The `UnRegisterAlias` command unregisters the specified subprocess alias from the current engine's registration database. For more information on aliases, see ["About Aliases" on page 143](#).

```
UnRegisterAlias alias_name t
```

Argument	Description
<i>alias_name</i>	The name of the alias (same as the process name).

The `UnRegisterAlias` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

Use the `ListRegistrations` Script command to display a list of components registered with the current engine (you can filter the list for aliases). The `UnRegisterAlias` command will remove the alias from the registration table, however it should be replaced by a new alias if the location of a subprocess is required by a process executing on the current engine.

UnRegisterAssignmentRules

The `UnRegisterAssignmentRules` command deletes all assignment rules (or just the specified rule) in the specified assignment rule dictionary from the current engine's registration database.

```
UnRegisterAssignmentRules dictionary_name [rule_name]
```

Argument	Description
<i>dictionary_name</i>	The name of the assignment rule dictionary to be unregistered.

Argument	Description
<i>rule_name</i>	The name of the assignment rule to be unregistered. If not specified, then all assignment rules in the specified dictionary will be unregistered.

The `UnRegisterAssignmentRules` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

Use the `ListRegistrations` Script command to display a list of components registered with the current engine (you can filter the list for assignment rule dictionaries). The `UnRegisterAssignmentRules` command removes the specified assignment rule or the entire assignment rule dictionary from the registration table, preventing subsequent use of any of the assignment rules in process execution on the current engine.

UnRegisterProcessDefinition

The `UnRegisterProcessDefinition` command deletes the specified process definition from the current engine's registration database.

`UnRegisterProcessDefinition process_name`

Argument	Description
<i>process_name</i>	The name of the registered process definition.

The `UnRegisterProcessDefinition` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

Use the `ListRegistrations` Script command to display a list of components registered with the current engine. You can filter the list for process definitions. The `UnRegisterProcessDefinition` command removes the specified process definition from the registration table but will have no effect on currently running process instances. It prevents subsequent creation of instances of the process definition on the current engine.

UnRegisterUserProfile

The `UnRegisterUserProfile` command deletes the specified user profile from the current engine's registration database.

```
UnRegisterUserProfile user_profile_name
```

Argument	Description
<i>user_profile_name</i>	The name of the registered user profile.

The `UnRegisterUserProfile` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

Use the `ListRegistrations` Script command to display a list of components registered with the current engine. You can filter the list for user profiles. The `UnRegisterUserProfile` command removes the specified user profile from the registration table.

WaitForStartup

The `WaitForStartup` command waits a specified timeout period for a specified engine unit to start up, and prints an error message if the engine unit does not start within the specified time.

```
WaitForStartup unit_name timeout
```

Argument	Description
<i>unit_name</i>	The specified engine unit.
<i>timeout</i>	A timeout period (in seconds) within which the specified engine unit must start up, or an error message is written.

The `WaitForStartup` command is available only when an engine is current, so you must first use the `FindEngine` Script command to set the current engine.

In a script, the `WaitForStartup` command is used after a `StartUnit` Script command. The `WaitForStartup` command causes the script to pause until the specified engine unit successfully starts up before continuing with subsequent Script commands. If the engine unit does not start up within the specified time, an error message is generated and the script aborts.

Engine Database Schema

This appendix documents the engine database schema. It lists the database tables and the specification of each, according to the following categories:

- current state tables
- registration tables
- history log tables

The tables are grouped first by category, and then alphabetically within each category. For an alphabetical listing of all tables, see [“Alphabetical Listing of Tables” on page 329](#).

Database Tables by Category

The tables comprising the engine database are listed in the following table. They are grouped into the following categories: current state, registration, and history log tables.

Current State Tables	Registration Tables	History Log Tables
	WFHRegistration	WFHRegistrationLog
	WFHAlias	WFHAliasLog
WFHProcess		WFHProcessLog* WFHProcessStateLog*
WFHProcAttributes		WFHProcAttribLog*
WFHLocks		WFHAttribLockLog*

* corresponds to History log configuration options. See [“How to Configure a New Engine” on page 95](#).

Current State Tables	Registration Tables	History Log Tables
WFHProcParameters		
WFHTerminatedProcs		
WFHTermProcParas		
WFHActivity		WFHActivityLog* WFHActStateLog*
WFHQueuedActivity		
WFHRoleBasedAct		
WFHActivityTokens		
WFHActivityUsers		
WFHAssignment		
WFHSession		WFHSessionLog*
WFHSessionRole		
WFHSessAttributes		
WFHRetryInfo		
WFHTimer		WFHTimerLog* WFHTimerStateLog*
WFHActiveTimer		
WFHComplexRule		
WFHControl		
WFHRecoveryHelp		
WFHColdStart		WFHColdStartLog WFHPerformanceLog WFHAlarmLog

* corresponds to History log configuration options. See ["How to Configure a New Engine"](#) on page 95.

Alphabetical Listing of Tables

The following is an alphabetical list of all tables created in the engine database, with a page reference to details about each.

Table B-1 Engine Database Tables

Database Table	See...	Description
WFHActiveTimer	page 332	Active timers that are not in memory.
WFHActivity	page 332	Activity instances that are current in the engine.
WFHActivityLog	page 348	Log of activity instances (both current and past) in the system.
WFHActivityTokens	page 335	Activity token counts—number of times the activity has completed—that are current in the system.
WFHActivityUsers	page 335	Activity link information: who has completed some activity. A row is held representing each activity that is listed as the activity link in some other activity.
WFHActStateLog	page 348	Log of state changes in activity instances in the engine.
WFHAlarmLog	page 349	A log of all alarms generated on this engine, recorded whenever history logging is turned on.
WFHAlias	page 346	Aliases currently registered with the engine.
WFHAliasLog	page 349	A log of aliases registered with the engine.
WFHAssignment	page 336	Used by the engine to track the stack of assignment rules generated by ConsultActivity and DelegateActivity method calls from client applications.
WFHAttribLockLog	page 350	Log of access obtained on all instances of process attributes.
WFHColdStart	page 336	System maintained table used by the engine before performing a cold start.
WFHColdStartLog	page 350	Always contains one row showing the last time the engine was cold started (thus dropping and re-creating all registration, state, and log tables).
WFHComplexRule	page 337	Used by the engine to track activities that contain complex assignment rules (assignment rules with custom Evaluate methods).

Table B-1 Engine Database Tables (*Continued*)

Database Table	See...	Description
WFHControl	page 337	System maintained table of sequence values for producing unique IDs for process, session, and timer instances.
WFHLocks	page 337	The type of access held on all instances of process attributes.
WFHPerformanceLog	page 351	Provides performance statistics for the engine. Average time measurements are expressed in milliseconds.
WFHProcAttribLog	page 352	Log of value changes to process attributes.
WFHProcAttributes	page 338	Current values of process attributes within all current process instances.
WFHProcess	page 339	All current process instances within the engine.
WFHProcessLog	page 353	Log of process instances created in this engine.
WFHProcessStateLog	page 354	Log of state changes to process instances in this engine.
WFHProcParameters	page 341	For a synchronous subprocess, the list of output parameters it must supply to its parent.
WFHQueuedActivity	page 341	Managed by the engine to maintain queued activities not in memory.
WFHRecoveryHelp	page 341	Internal use only.
WFHRegistration	page 347	Currently registered iIS plans.
WFHRegistrationLog	page 354	Log of registrations with the engine.
WFHRetryInfo	page 342	Internal use only. Managed by the engine to ensure consistency between engine actions and messages to client applications.
WFHRoleBasedAct	page 342	Internal use only. Managed by the engine to manage role-based activities.
WFHSessAttributes	page 343	User profile attributes associated with a session.
WFHSession	page 343	Currently active and suspended client sessions with the engine.
WFHSessionLog	page 355	Log of current and past client session.

Table B-1 Engine Database Tables (*Continued*)

Database Table	See...	Description
WFHSessionRole	page 344	Current client sessions and their roles. If a session has multiple roles, this table has a separate row for each role.
WFHTerminatedProcs	page 344	Internal use only. Managed by the engine to track aborted synchronous subprocesses.
WFHTermProcParas	page 345	Internal use only. Managed by the engine to track output parameters from aborted synchronous subprocesses.
WFHTimer	page 345	Timer instances currently instantiated in the engine.
WFHTimerLog	page 355	Log of timer instances that have been instantiated in the engine.
WFHTimerStateLog	page 356	Log of the state of timer instances for the engine.

Database Schema Reference

This section provides an annotated description of all the tables created and maintained by an iIS process engine. The iIS process engine automatically creates these tables when the engine is cold started and maintain their contents during normal operations. You can use the information provided here to write queries against the tables for management information on current and past process instances. However, you should avoid altering the rows in the tables (with the exception of periodically flushing history tables of rows that are no longer needed).

Some tables are for internal use only, and are unlikely to be useful for any management queries. The documentation on these tables is limited.

The data type description for each entry in a table is based on the representation used in Oracle databases. The actual data type may differ, depending on the database vendor you are using.

NOTE The schema presented here may differ from the schema for previous versions of iIS. Also, the schema is subject to further change in future versions.

Current State Tables

The following is an alphabetical list of all current state tables in the engine database. Indexes to tables are indicated by a checkmark (✓) in the Index column. In some cases, a table has an alternate index, indicated by a bullet (●).

For a description of each of these tables, refer to [Table B-1 on page 329](#).

Table B-2 WFHActiveTimer

Name	Index	Null?	Type	Description
ID	✓	NOT NULL	NUMBER(38)	Identifier for the timer instance within the process instance designated by PROCESSID.
PROCESSID	✓	NOT NULL	NUMBER(38)	Unique identifier for the process instance in which this active timer occurs.
TIMETOALARM			DATE	Date time stamp indicating when the timer is expected to expire.

Table B-3 WFHActivity

Name	Index	Null?	Type	Description
PROCESSID	✓	NOT NULL	NUMBER(38)	Unique identifier for the process instance in which this activity instance occurs.
ROOTPROCID			NUMBER(38)	Identifier for the process instance on this engine that originally invoked this process. For a process instance that is not running as a subprocess, this is the same as PROCESSID. For a subprocess, this is the PROCESSID of the parent process instance, or its parent, or its parent; up to the original initiating process instance.

Table B-3 WFHActivity (Continued)

Name	Index	Null?	Type	Description
TYPE			NUMBER(38)	Type of activity: 1 – First activity 2 – Last activity 3 – Offered activity 4 – Subprocess activity 5 – Automatic activity 6 – Queued activity 7 – Junction activity Note: These constant values are internal to the engine only. You should avoid their use if possible.
ID	✓	NOT NULL	NUMBER(38)	Unique identifier for this activity instance within the process instance in which it occurs.
STATE			NUMBER(38)	Activity state: 10 – WFActivity.PENDING 20 – WFActivity.READY 25 – WFActivity.PREPARE_ACTIVE 30 – WFActivity.ACTIVE 35 – WFActivity.PREPARE_COMPLETE 40 – WFActivity.COMPLETED 45 – WFActivity.PREPARE_ABORT 50 – WFActivity.ABORTED
PREVSTATE			NUMBER(38)	The state the activity was in immediately prior to the current state. See the values for STATE above.
CREATETIME			DATE	Time that the activity instance was created.
SESSIONID	●		NUMBER(38)	For an ACTIVE activity, the ID of the session that owns the activity. This column links to column WFHSESSION.ID

Table B-3 WFHActivity (Continued)

Name	Index	Null?	Type	Description
ACTNAME			VARCHAR2(64)	The name of the activity in the process definition.
ABORTACTION			NUMBER(38)	Abort routing information. Internal use only.
WUSER			VARCHAR2(64)	The "LinkedUser" information passed into this activity by its activity link. If there is no activity link, value is "<NO_LINKED_USER>"
OTHERINFO			VARCHAR2(64)	The "OtherInfo" information passed into this activity by its activity link. If there is no activity link, value is "<NO_OTHERINFO>"
CURASSIGNRULE			VARCHAR2(64)	This column is used internally by the engine to track dynamic assignment rules during a ConsultActivity or DelegateActivity method call from a client application.
NEXTASSIGNID			NUMBER(38)	This column is used internally by the engine to track dynamic assignment rules during a ConsultActivity or DelegateActivity method call from a client application.
FIFOORDER			NUMBER(38)	For a queued activity with the queue prioritized by 'First In First Out' (the default), a system-assigned sequence value for placement of this activity in the queue.
SUSPENDACTION			NUMBER(38)	Actions to take when the activity is suspended: 700 – WFActivity.RETAIN 701 – WFActivity.REMOVE
MARKER		NOT NULL	NUMBER(38)	Internal use only.
PRIORITY		NOT NULL	NUMBER(38)	Internal use only.
READYCOUNT		NOT NULL	NUMBER(38)	Internal use only.

Table B-4 WFHActivityTokens

Name	Index	Null?	Type	Description
PROCESSID	✓	NOT NULL	NUMBER(38)	Unique identifier for the process instance in which this activity instance occurs.
ACTIVITYID	✓	NOT NULL	NUMBER(38)	Unique identifier for a pending activity instance within the process instance in which it occurs.
ACTIVITYNAME			VARCHAR2(64)	The name of an activity that is an immediate predecessor of the activity instance identified by the ACTIVITYID in this row.
TOKCOUNTER			NUMBER(38)	Value of the attribute <code>_COUNT<ActivityName></code> , where <code><ActivityName></code> is the same as ACTIVITYNAME in this row.

Table B-5 WFHActivityUsers

Name	Index	Null?	Type	Description
PROCESSID	✓	NOT NULL	NUMBER(38)	Unique identifier for the process instance in which this activity instance occurs.
ACTIVITYNAME	✓	NOT NULL	VARCHAR2(64)	The name of the activity in the process definition.
WUSER			VARCHAR2(64)	The name of the user who completed the last instance of this activity.
OTHERINFO			VARCHAR2(64)	The otherInfo of the user who completed the last instance of this activity.

Table B-6 WFHAssignment

Name	Index	Null?	Type	Description
PROCESSID	✓	NOT NULL	NUMBER(38)	Unique identifier for the process instance in which this activity instance occurs.
ACTIVITYID	✓	NOT NULL	NUMBER(38)	Identifier for the activity instance in the process instance designated by PROCESSID.
ID	✓		NUMBER(38)	Identifier for the assignment rule.
LINKEDUSER			VARCHAR2(64)	The "LinkedUser" information passed into this activity by its activity link. If there is no activity link, value is "<NO_LINKED_USER>"
OTHERINFO			VARCHAR2(64)	The "OtherInfo" information passed into this activity by its activity link. If there is no activity link, value is "<NO_OTHERINFO>"
ASSIGNMENTRULE			VARCHAR2(64)	The name of an assignment rule generated by a call to ConsultActivity or DelegateActivity from a client application.

Table B-7 WFHColdStart

Name	Index	Null?	Type	Description
SEQID			NUMBER(38)	Internal use only.
VERNUM			NUMBER(38)	Schema version of the engine database.
LOGVER			NUMBER(38)	Version of the database log table.
REGVER			NUMBER(38)	Version of the database registration table.
STATEVER			NUMBER(38)	Version of the database state table.

Table B-8 WFHComplexRule

Name	Index	Null?	Type	Description
PROCESSID	✓		NUMBER(38)	Unique identifier for the process instance in which this activity instance occurs.
ACTIVITYID	✓		NUMBER(38)	Identifier for the activity instance in the process instance designated by PROCESSID.

Table B-9 WFHControl

Name	Index	Null?	Type	Description
NEXTSESSIONID			NUMBER(38)	System maintained sequence for session ID values.
NEXTPROCESSID			NUMBER(38)	System maintained sequence for process ID values.
NEXTTIMERID			NUMBER(38)	System maintained sequence for timer ID values.
HISTORYDETAIL			NUMBER(38)	Internal use only.

Table B-10 WFHLocks

Name	Index	Null?	Type	Description
PROCESSID	✓		NUMBER(38)	Unique identifier for the process instance in which this process attribute occurs.
ATTNAME	✓		VARCHAR2(64)	Name of the process attribute.
ACTIVITYID			NUMBER(38)	If the access to the process attribute is associated with an activity, the unique identifier for this activity instance.
SESSIONID			NUMBER(38)	If the access to the process attribute is associated with a user session, the unique identifier for this session.

Table B-10 WFHLocks (Continued)

Name	Index	Null?	Type	Description
LOCKTYPE			NUMBER(38)	Type of access to the process attribute: 1 – WFAttributeAccessor.READ 2 – WFAttributeAccessor.READQ 3 – WFAttributeAccessor.WRITE 4 – WFAttributeAccessor.WRITEQ 5 – WFAttributeAccessor.NO_LOCK
ACCESSORID	✓		NUMBER(38)	Unique identifier for the attribute accessor instance within the process instance.

Table B-11 WFHProcAttributes

Name	Index	Null?	Type	Description
PROCESSID		NOT NULL	NUMBER(38)	Unique identifier for the process instance in which this process attribute occurs.
ATTNAME			VARCHAR2(64)	Name of the process attribute.
ATTTYPE			NUMBER(38)	Type of the attribute: 1 – IntegerData 2 – TextData 3 – BooleanData 4 – DateTimeData 5 – DoubleData 6 – IntervalData 7 – DecimalData 8– XMLData
ATTSEGMENTID	✓		NUMBER(38)	Internal use only.
ATTSEGLENGTH			NUMBER(38)	Internal use only.
ATTVALUE			VARCHAR2(255)	The value of the attribute, in readable character form.

Table B-12 WFHPProcess

Name	Index	Null?	Type	Description
ID	✓	NOT NULL	NUMBER(38)	Unique identifier for the process instance in this engine.
TYPE			NUMBER(38)	Type of process (means of process creation): 1 – Process 2 – Subprocess
STATE			NUMBER(38)	State of the process instance: 405 – WFProcess.PREPARE_CREATE 410 – WFProcess.CREATED 420 – WFProcess.ACTIVE 430 – WFProcess.COMPLETED 435 – WFProcess.PREPARE_ABORT 440 – WFProcess.ABORTED
SUBMITTIME			DATE	Date and time the process instance was begun.
PARENTACTIVITY			NUMBER(38)	For a subprocess, the activity ID of the subprocess activity instance in the parent process that invoked this process.
PARENTPROCESSID			NUMBER(38)	For a subprocess, the process ID for the process instance that started this process instance.
ROOTPROCESSID			NUMBER(38)	For a subprocess, the process ID for the process instance that ultimately started this process (the parent of the parent of the parent ... of this process, up to the original process that is not a subprocess).
PARENTENVIRONMENT			VARCHAR2(64)	For a subprocess, the environment of the engine of the process instance that started this process instance.
PARENTENGINE			VARCHAR2(64)	For a subprocess, the name of the engine of the process instance that started this process.
COLDSEQID			NUMBER(38)	Internal use only.

Table B-12 WFHPProcess (Continued)

Name	Index	Null?	Type	Description
NEXTACTIVITYID			NUMBER(38)	Sequence for the next activity instance within this process instance.
INTERNALNAME			VARCHAR2(138)	Internal use only.
PROCNAME			VARCHAR2(64)	The name of the process definition of which this running process is an instance.
RECOVERYLEVEL		NOT NULL	NUMBER(38)	The recovery level designated for this process: 0 – WFProcess.RCVR_NORMAL 501 – WFProcess.RCVR_ONLY 502 – WFProcess.RCVR_NONE 503 – WFProcess.RCVR_FULL 504 – WFProcess.RCVR_PARENT
PARENTACTNAME			VARCHAR2(64)	For a subprocess, the name of the parent activity that invoked the subprocess.
USERID			VARCHAR2(64)	The user who started the process.
PATTNAME			VARCHAR2(64)	The name of the primary attribute of the process.
COMPLEVEL		NOT NULL	NUMBER(38)	Compatibility level for the process.
MSTATUS		NOT NULL	NUMBER(38)	Internal use only.

Table B-13 WFHProcParameters

Name	Index	Null?	Type	Description
PROCESSID	✓	NOT NULL	NUMBER(38)	The PROCESSID of a synchronous subprocess that has an output parameter.
ATTNAME			VARCHAR2(64)	The name of a process attribute that must be returned to the calling process (an output parameter).

Table B-14 WFHQueuedActivity

Name	Index	Null?	Type	Description
ID		NOT NULL	NUMBER(38)	Identifier for the activity being queued.
PROCESSID	●	NOT NULL	NUMBER(38)	Unique identifier for the process instance containing the queued activity.
FIFOORDER	✓	NOT NULL	NUMBER(38)	The placement of the activity in the queue, based on FIFO order (first in, first out).
PRIORITY	✓	NOT NULL	NUMBER(38)	The priority of the queued activity.
QUEUENAME	✓	NOT NULL	VARCHAR2(129)	The name of the activity queue in which the activity is placed.

Table B-15 WFHRecoveryHelp

Name	Index	Null?	Type	Description
PROCESSID		NOT NULL	NUMBER(38)	Unique identifier for a process instance in the engine.
ACTIVITYID		NOT NULL	NUMBER(38)	Identifier for an activity instance in the process instance designated by PROCESSID.

Table B-16 WFHRetryInfo

Name	Index	Null?	Type	Description
GROUPID	✓	NOT NULL	NUMBER(38)	Identifier for the session between the engine and client.
SEQNUMBER		NOT NULL	NUMBER(38)	Identifier for the retry attempt.
PROCESSID			NUMBER(38)	Unique identifier for a process instance in the engine.
ACTIVITYID			NUMBER(38)	Identifier for an activity instance in the process instance designated by PROCESSID.
REQSTATUS			NUMBER(38)	Internal use only.
ACTIONFLAG			NUMBER(38)	Internal use only.

Table B-17 WFHRoleBasedAct

Name	Index	Null?	Type	Description
<i>PROCESSID</i>	●	NOT NULL	NUMBER(38)	Unique identifier for a process instance in the engine.
ID	✓	NOT NULL	NUMBER(38)	Identifier for the role-based activity instance in the process instance designated by PROCESSID.
ROLENAME	✓		VARCHAR2(64)	The user role name.
STATE		NOT NULL	NUMBER(38)	Internal use only.

Table B-18 WFHSessAttributes

Name	Index	Null?	Type	Description
SESSIONID	✓	NOT NULL	NUMBER(38)	Identifier for the session between the engine and client application.
ATTNAME		NOT NULL	VARCHAR2(64)	Name of a process attribute in the session.
ATTTYPE		NOT NULL	NUMBER(38)	The type of the process attribute.
ATTSEGMENTID		NOT NULL	NUMBER(38)	Internal use only.
ATTSEGLENGTH		NOT NULL	NUMBER(38)	Internal use only.
ATTVALUE		NOT NULL	VARCHAR2(255)	The value of the attribute (in text format).
SETNUMBER	✓	NOT NULL	NUMBER(38)	Internal use only.

Table B-19 WFHSession

Name	Index	Null?	Type	Description
ID	✓	NOT NULL	NUMBER(38)	Unique identifier for this client session.
SESSNAME			VARCHAR2(64)	The session name.
USERNAME			VARCHAR2(64)	The user name of the user on the session.
PROFILENAME			VARCHAR2(64)	The profile name of the user on the session.
STATE			NUMBER(38)	The state of the session: 210 – WFSession.ACTIVE 220 – WFSession.SUSPENDED
SESSMODE			NUMBER(38)	Internal use only.
SESSTXID			VARCHAR2(255)	The transaction ID, used only in iIS two-phase commit operations.
OTHERINFO			VARCHAR2(64)	The otherInfo attribute associated with a user profile.

Table B-20 WFHSessionRole

Name	Index	Null?	Type	Description
SESSIONID		NOT NULL	NUMBER(38)	Unique identifier for the client session. Joins to WFHSESSION.ID.
ROLENAME	●		VARCHAR2(64)	A role held by the session.

Table B-21 WFHTerminatedProcs

Name	Index	Null?	Type	Description
COMBINEDID	✓	NOT NULL	VARCHAR2(158)	Internal use only.
PROCESSID			NUMBER(38)	Unique identifier for a process instance in the engine.
ACTIVITYID			NUMBER(38)	Identifier for an activity instance within the process instance designated by PROCESSID.
ENVIRONMENT			VARCHAR2(64)	The name of the environment the engine is running in.
ENGINE			VARCHAR2(64)	The name of the engine.
COLDSEQID			NUMBER(38)	Internal use only.
COMPLETIONCODE			NUMBER(38)	Indicates the completion state of the process: 440 – Aborted 430 – Completed
USERNAME			VARCHAR2(64)	The user name for the user that started the process.
OTHERINFO			VARCHAR2(64)	The otherInfo attribute associated with a user profile.

Table B-22 WFHTermProcParas

Name	Index	Null?	Type	Description
COMBINEDID	✓	NOT NULL	VARCHAR2(158)	Internal use only.
ATTNAME			VARCHAR2(64)	The name of the attribute that must be returned to the calling process (an output parameter).
ATTTYPE			NUMBER(38)	Attribute type.
ATTSEGMENTID			NUMBER(38)	Internal use only.
ATTSEGLENGTH			NUMBER(38)	Internal use only.
ATTVALUE			VARCHAR2(255)	The value of the attribute (in text format)

Table B-23 WFHTimer

Name	Index	Null?	Type	Description
ID	✓	NOT NULL	NUMBER(38)	Unique identifier for the timer instance within this engine.
PROCESSID	✓	NOT NULL	NUMBER(38)	Unique identifier for the process instance in which this timer instance occurs.
TIMERNAME			VARCHAR2(64)	Name of the timer in the process definition.
STATE			NUMBER(38)	State of the timer: 820 – WFHTimer.OFF 830 – WFHTimer.ON
RELATIVETIME			VARCHAR2(32)	For an elapsed timer, the defined duration of time set for the timer, expressed in character form. For a deadline timer, the defined expiration time set for the timer, expressed in character form.

Table B-23 WFHTimer (Continued)

Name	Index	Null?	Type	Description
CURRELATIVETIME			VARCHAR2(32)	For a paused elapsed timer, the duration remaining should the timer be turned on again without reset, expressed in character form. For a paused deadline timer, the “remembered” expiration time should the timer be turned on again without reset, expressed in character form.
TIMETOALARM			DATE	For a running timer, the time when it will expire.
TYPE			NUMBER(38)	Type of timer: 1 – WFHTimer.ELAPSED 2 – WFHTimer.DEADLINE

Registration Tables

This section contains an alphabetical list of all registration tables in the engine database. For a description of each of these tables, refer to [Table B-1 on page 329](#).

Table B-24 WFHAlias

Name	Index	Null?	Type	Description
ALIAS	✓	NOT NULL	VARCHAR2(64)	Name of an alias registered on this engine.
PROCESS			VARCHAR2(64)	Name of the process definition which the alias represents (same as the alias name).
ENGINE			VARCHAR2(64)	Engine where the actual process definition resides.
ENVIRONMENT			VARCHAR2(64)	Environment of the engine where the actual process definition resides.

Table B-25 WFHRegistration

Name	Index	Null?	Type	Description
APPLICATIONID	✓	NOT NULL	VARCHAR2(32)	Name of the installed application for this iIS plan.
COMPATIBILITYLEVEL	✓	NOT NULL	NUMBER(38)	Compatibility level of the installed library of this iIS plan.
LIBRARYNAME			VARCHAR2(64)	Name of the installed library for this iIS plan.
LOADOPTIONS			NUMBER(38)	Internal use only.
PROJECTNAME			VARCHAR2(67)	Name of the generated TOOL project of the compiled iIS plan.
REGISTRATIONNAME	✓	NOT NULL	VARCHAR2(64)	Name of the source iIS plan from which the iIS component is derived.
TYPE	✓	NOT NULL	NUMBER(38)	Type of iIS plan: 151-Process Definition 152-Validation 153-Assignment Rule 154-User Profile
WUSAGE			NUMBER(38)	Internal use only.
CREATETIME			DATE	Date and time of registration.
OBSOLETE	✓	NOT NULL	NUMBER(38)	Internal use only.

History Log Tables

The following is an alphabetical list of all history log tables in the engine database. “[State Values](#)” on [page 357](#) lists the possible values for the State field present in many of the history log tables.

For a description of each of these tables, refer to [Table B-1 on page 329](#).

Table B-26 WFHActivityLog

Name	Index	Null?	Type	Description
ID			NUMBER(38)	Unique identifier for this activity instance within the process instance in which it occurs.
ACTNAME			VARCHAR2(64)	The name of the activity in the process definition.
CREATETIME			DATE	Time that the activity instance was created.
PROCESSID			NUMBER(38)	Unique identifier for the process instance in which this activity instance occurs.
LOGTIME			DATE	Time of this logging.

Table B-27 WFHActStateLog

Name	Index	Null?	Type	Description
ID			NUMBER(38)	Unique identifier for this activity instance within the process instance in which it occurs.
PROCESSID			NUMBER(38)	Unique identifier for the process instance in which this activity instance occurs.
STATE			NUMBER(38)	Activity state: 10 – WFActivity.PENDING 20 – WFActivity.READY 25 – WFActivity.PREPARE_ACTIVE 30 – WFActivity.ACTIVE 35 – WFActivity.PREPARE_COMPLETE 40 – WFActivity.COMPLETED 45 – WFActivity.PREPARE_ABORT 50 – WFActivity.ABORTED
PREVSTATE			NUMBER(38)	Immediately previous state of this activity. Values are the same as in STATE above.

Table B-27 WFHActStateLog (Continued)

Name	Index	Null?	Type	Description
SESSIONID			NUMBER(38)	ID of the session that owns this activity (if the activity is ACTIVE).
LOGTIME			DATE	Time of this logging.

Table B-28 WFHAlarmLog

Name	Index	Null?	Type	Description
ALARMTYPE			VARCHAR2(64)	The type of the alarm.
DETAILS			VARCHAR2(255)	The text of the alarm.
LOGTIME			DATE	Time of this logging.

Table B-29 WFHAliasLog

Name	Index	Null?	Type	Description
ALIAS			VARCHAR2(64)	Name of an alias registered (or now unregistered) on this engine.
PROCESS			VARCHAR2(64)	Name of the process definition which the alias represents (same as the alias name).
ENGINE			VARCHAR2(64)	Engine where the actual process definition resides.
ENVIRONMENT			VARCHAR2(64)	Environment of the engine where the actual process definition resides.
STATE			NUMBER(38)	State of this alias registration: 1 – Registered -1 – Unregistered
LOGTIME			DATE	Time of this logging.

Table B-30 WFHAttribLockLog

Name	Index	Null?	Type	Description
PROCESSID			NUMBER(38)	Unique identifier for the process instance in which this process attribute occurs.
ACTIVITYID			NUMBER(38)	If the access to the process attribute is associated with an activity, the unique identifier for this activity instance.
SESSIONID			NUMBER(38)	If the access to the process attribute is associated with a user session, the unique identifier for this session.
ACCESSORID			NUMBER(38)	Unique identifier for this attribute accessor instance within this process instance.
ATTNAME			VARCHAR2(64)	Name of the process attribute.
LOCKTYPE			NUMBER(38)	Type of access to the process attribute: 0 – Not applicable (used internally) 1 – WFAttributeAccessor.READ 2 – WFAttributeAccessor.READQ 3 – WFAttributeAccessor.WRITE 4 – WFAttributeAccessor.WRITEQ 5 – WFAttributeAccessor.NO_LOCK
STATE			NUMBER(38)	Internal use only.
LOGTIME			DATE	Time of this logging.

Table B-31 WFHColdStartLog

Name	Index	Null?	Type	Description
SEQID			NUMBER(38)	Internal use only.
STARTTIME			DATE	Date and time this engine was last cold started.
PROCATTSIZE			NUMBER(38)	Internal use only.
SESSATTSIZE			NUMBER(38)	Internal use only.

Table B-32 WFHPerformanceLog

Name	Index	Null?	Type	Description
SAMPLETIME		NOT NULL	DATE	Date time stamp for the sample.
RESPONSETIME		NOT NULL	NUMBER(38)	Average engine response time to requests by a client application.
SWAPINTIME		NOT NULL	NUMBER(38)	Average process swap in time.
SWAPOUTTIME		NOT NULL	NUMBER(38)	Average process swap out time.
TXCOMMITTIME		NOT NULL	NUMBER(38)	Average engine transaction commit time.
DBSACQUIRETIME		NOT NULL	NUMBER(38)	Average DB service acquire time.
APPCPUTIME		NOT NULL	NUMBER(38)	Engine CPU time.
GCCPUTIME		NOT NULL	NUMBER(38)	Engine garbage collection time.
ACTIVESESSION		NOT NULL	NUMBER(38)	The number of active sessions.
SUSPENDSESSION		NOT NULL	NUMBER(38)	The number of suspended sessions.
ACTIVEPROCESS		NOT NULL	NUMBER(38)	The number of active processes.
ACTIVESUBPROCESS		NOT NULL	NUMBER(38)	The number of active subprocesses.
PENDINGACTIVITY		NOT NULL	NUMBER(38)	The number of pending activities.
READYACTIVITY		NOT NULL	NUMBER(38)	The number of ready activities.
ACTIVEACTIVITY		NOT NULL	NUMBER(38)	The number of active activities.
RUNNINGTIMER		NOT NULL	NUMBER(38)	The number of running timers.
STOPPEDTIMER		NOT NULL	NUMBER(38)	The number of stopped timers.

Table B-32 WFHPerformanceLog (Continued)

Name	Index	Null?	Type	Description
ACTIVEPAGE		NOT NULL	NUMBER(38)	The number of active memory pages for the engine.
ALLOCATEPAGE		NOT NULL	NUMBER(38)	The number of allocated memory pages for the engine.

Table B-33 WFHProcAttribLog

Name	Index	Null?	Type	Description
PROCESSID			NUMBER(38)	Unique identifier for the process instance in which this process attribute occurs.
ATTNAME			VARCHAR2(64)	Name of the process attribute.
ATTTYPE			NUMBER(38)	Type of the attribute: 1 – IntegerData 2 – TextData 3 – BooleanData 4 – DateTimeData 5 – DoubleData 6 – IntervalData 7 – DecimalData 8 – XmlData
ATTSEGMENTID			NUMBER(38)	Internal use only.
ATTSEGLENGTH			NUMBER(38)	Internal use only.
ATTVALUE			VARCHAR2(255)	New value of the attribute.
STATE			NUMBER(38)	New state of the process attribute: 1 – Instantiation 0 – Value change -1 – Uninstantiation
LOGTIME			DATE	Time of this logging.

Table B-34 WFHProcessLog

Name	Index	Null?	Type	Description
ID			NUMBER(38)	Unique identifier for the process instance in this engine.
PROCNAME			VARCHAR2(64)	The name of the process definition of which this process is an instance.
SUBMITTIME			DATE	Date and time the process instance was begun.
PARENTACTIVITY			NUMBER(38)	For a subprocess, the activity ID of the subprocess activity instance in the parent process that invoked this process. The value is 0 if this is not a subprocess.
PARENTPROCID			NUMBER(38)	For a subprocess, the process ID for the process instance that started this process instance. The value is 0 if this is not a subprocess.
ROOTPROCID			NUMBER(38)	For a subprocess, the process ID for the process instance that ultimately started this process (the parent of the parent of the parent ... of this process, up to the original process that is not a subprocess). The value is 0 if this is not a subprocess.
LOGTIME			DATE	Time of this logging.
PARENTENVIRONMENT			VARCHAR2(64)	The name of the parent environment for the parent process instance indicated by PARENTPROCID.
PARENTENGINE			VARCHAR2(64)	The name of the engine containing the parent process instance indicated by PARENTPROCID.

Table B-35 WFHProcessStateLog

Name	Index	Null?	Type	Description
ID			NUMBER(38)	Unique identifier for the process instance in this engine.
STATE			NUMBER(38)	New state of the process instance: 405 – WFProcess.PREPARE_CREATE 410 – WFProcess.CREATED 420 – WFProcess.ACTIVE 430 – WFProcess.COMPLETED 435 – WFProcess.PREPARE_ABORT 440 – WFProcess.ABORTED
LOGTIME			DATE	Time of this logging.

Table B-36 WFHRegistrationLog

Name	Index	Null?	Type	Description
APPLICATIONID			VARCHAR2(32)	Name of the installed application for this iIS plan.
COMPATIBILITYLEVEL			NUMBER(38)	Compatibility level of the installed library of this iIS plan.
PROJECTNAME			VARCHAR2(67)	Name of the generated TOOL project of the compiled iIS plan.
REGISTRATIONNAME			VARCHAR2(64)	Name of the source iIS plan from which the iIS component is derived.
TYPE			NUMBER(38)	Type of iIS plan: 151-Process Definition 152-Validation 153-Assignment Rule 154-User Profile
OBSOLETE			NUMBER(38)	Internal use only.

Table B-36 WFHRegistrationLog (Continued)

Name	Index	Null?	Type	Description
STATE			NUMBER(38)	Effect of this registration action: 1 – Registered -1 – Unregistered
LOGTIME			DATE	Time of this logging.

Table B-37 WFHSessionLog

Name	Index	Null?	Type	Description
ID			NUMBER(38)	Unique identifier for the client session.
SESSNAME			VARCHAR2(64)	The session name (not logged on session termination).
USERNAME			VARCHAR2(64)	The user name for the session user (not logged on session termination).
PROFILENAME			VARCHAR2(64)	The profile name of the user on the session.
STATE			NUMBER(38)	Logged new state for the session: 210 – WFSession.ACTIVE 220 – WFSession.SUSPENDED 230 – WFSession.TERMINATED
LOGTIME			DATE	Time of this logging.

Table B-38 WFHTimerLog

Name	Index	Null?	Type	Description
ID			NUMBER(38)	Unique identifier for the timer instance within this engine.
PROCESSID			NUMBER(38)	Unique identifier for the process instance in which this timer instance occurs.
TIMERNAME			VARCHAR2(64)	Name of the timer in the process definition.
LOGTIME			DATE	Time of this logging.

Table B-39 WFHTimerStateLog

Name	Index	Null?	Type	Description
ID			NUMBER(38)	Unique identifier for the timer instance within this engine.
PROCESSID			NUMBER(38)	Unique identifier for the process instance in which this timer instance occurs.
STATE			NUMBER(38)	New state of the timer: 810 – WFTimer.CREATED 820 – WFTimer.OFF 830 – WFTimer.ON 840 – WFTimer.EXPIRED 850 – WFTimer.DELETED
RELATIVETIME			VARCHAR2(32)	For an elapsed timer, the defined duration of time set for the timer, expressed in character form. For a deadline timer, the defined expiration time set for the timer, expressed in character form.
CURRELATIVETIME			VARCHAR2(32)	For a paused elapsed timer, the duration remaining should the timer be turned on again without reset, expressed in character form. For a paused deadline timer, the “remembered” expiration time should the timer be turned on again without reset, expressed in character form.
TIMETOALARM			DATE	For a running timer, the time when it is/was to expire.
LOGTIME			DATE	Time of this logging.

State Values

Many of the history log tables listed in this section have a field that represents the state of a given engine object: process instance, activity, session, and timer. The following table shows the values corresponding to the various states of these objects:

Object	State	Value
Process Instance	PREPARE_CREATE	405
	CREATED	410
	ACTIVE	420
	COMPLETED	430
	PREPARE_ABORT	435
	ABORTED	440
Activity	PENDING	10
	READY	20
	PREPARE_ACTIVE	25
	ACTIVE	30
	PREPARE_COMPLETE	35
	COMPLETED	40
	PREPARE_ABORT	45
	ABORTED	50
Timer	DELETED	60
	CREATED	810
	OFF	820
	ON	830
	EXPIRED	840
Session	DELETED	850
	ACTIVE	210
	SUSPENDED	220
	TERMINATED	230

Index

A

- AbortActivity command, Conductor Script 273
- AbortAllProcesses command, Conductor Script 274
- ABORTED
 - activity state 163, 199, 357
 - process instance state 173, 198, 357
- aborting
 - activities 186
 - process instances 181
- AbortProcess command, Conductor Script 274
- ACTIVE
 - activity state 163, 199, 357
 - process instance state 198, 357
 - session state 158, 199, 357
- activities
 - about 162
 - creation 167
 - methods, execution of 164
 - state transitions 168
 - status, checking 185
 - termination 171
- activity execution
 - automatic activities 171
 - offered activities 168
 - queued activities 169
 - subprocess activities 170
- activity queues
 - reprioritizing 189
 - status, checking 188
- activity states
 - ABORTED 163, 199, 357
 - ACTIVE 163, 199, 357
 - changing 186
 - COMPLETED 163, 199, 357
 - DELETED 199, 357
 - PENDING 162, 199, 357
 - process execution and 167
 - READY 162, 199, 357
 - transitions 164, 168
- activity types 165
- alarms
 - about 204
 - list and descriptions 205
 - monitoring 206
 - viewing 208
- Alarms window 205
- aliases
 - about 143
 - registration 144, 148
 - subprocess activity and 143
 - unregistration 150
- Assignment Rule dictionary, registration 144
- audit trace 211
- automatic activities
 - described 165
 - execution of 171
- auto-reconnect mechanism, sessions 159

B

- backup engine unit
 - defined 33
 - partitioning guidelines 86
- BOTH governor state 106
- BroadcastMessage command, Conductor Script 275

C

- cconsole command 67
- central server node
 - definition 51
 - installation 57
- ChannelDisconnect alarm 205
- client application development node 52
- client applications 45
- command summary
 - Conductor Script 261
 - iIS Console 78
- CommitTransaction command, Conductor Script 275
- CompleteActivity command, Conductor Script 276
- COMPLETED
 - activity state 163, 199, 357
 - process instance state 173, 198, 357
- component mode Conductor Script commands 271
- Conductor Script
 - about 41
 - command summary 261
 - component mode commands 271
 - engine management commands 264
 - engine status, showing 309
 - environment mode commands 263
 - iIS distributions 144
 - process execution management commands 266
 - starting 228
- Conductor Script command
 - AbortActivity 273
 - AbortAllProcesses 274
 - AbortProcess 274
 - BroadcastMessage 275
 - CommitTransaction 275
 - CompleteActivity 276
 - ConsultActivity 276
 - CreateActivity 278
 - CreateFilter 279
 - DelegateActivity 281
 - DeleteFilter 283
 - FindDBService 283
 - FindEngine 284
 - FindGovernor 284
 - FindNode 285
 - FindParentEngine 285
 - FindPrimary 285
 - FindUnit 286
 - FlushLog 286
 - IIOPServer 287
 - ListActivities 287
 - ListActivityQueues 287
 - ListConductorDistributions 288
 - ListEngines 289
 - ListFilters 289
 - ListProcesses 289
 - ListRegistrations 290
 - ListSessions 291
 - ListTimers 291
 - ListTransactions 291
 - MakeConductorDistribution 292
 - ModLogFlags 292
 - ReadyActivity 293
 - RegisterAlias 294
 - RegisterAssignmentRules 294
 - RegisterProcessDefinition 295
 - RegisterUserProfile 295
 - RegisterValidation 296
 - RemoveReadLock 297
 - RemoveWriteLock 297
 - Rollback Transaction 298
 - RollbackActivity 298
 - SendMessage 299
 - SetAttributeValue 300
 - SetPassword 300
 - SetPrimary 301
 - SetQueuedActivityPriority 301
 - SetState 302
 - SetTimer 302
 - SetTimerDeadline 303
 - SetTimerElapsed 304
 - ShowActivity 304

Conductor Script command (*continued*)

- ShowActivityQueue 305
 - ShowConfiguration 306
 - ShowEngine 306
 - ShowLogFlags 307
 - ShowProcess 307
 - ShowSession 308
 - ShowStatus 309
 - ShowTimer 313
 - Shutdown 314
 - StartActivity 314
 - StartDBService 315
 - StartEngine 315
 - StartGovernor 316
 - StartTimer 317
 - StartUnit 317
 - StopTimer 319
 - SuspendAllSessions 319
 - SuspendSession 320
 - TerminateAllSessions 320
 - TerminateSession 321
 - Uninstall 321
 - UnRegisterAlias 322
 - UnRegisterAssignmentRules 322
 - UnRegisterProcessDefinition 323
 - UnRegisterUserProfile 324
 - WaitForStartup 324
- configuration file, engine 90
- ConsultActivity command, Conductor Script 276
- CreateActivity command, Conductor Script 278
- CREATED states 198, 357
- CreateFilter command, Conductor Script 279
- Cscript, *See* Conductor Script
- current state tables
 - engine database 124, 327
 - schema 332

D

- data file, dump/restore facility 128
- database
 - See also* engine database
 - Dump/Restore facilities 126
 - engine 32

- enterprise 45
 - organization 46
- database service
 - configuring 98, 99
 - defined 32
 - monitoring 121
 - partitioning guidelines 87
 - startup 109
- database service state 109
- DatabaseDisconnect alarm 205
- DelegateActivity command, Conductor Script 281
- DELETED states 199, 357
- DeleteFilter command, Conductor Script 283
- development client node, installation 59
- development engines 83
- development repository 45
- disconnect parameters, sessions 160
- distributions
 - iIS 144
 - making with Conductor Script 243
- DrDump command, Conductor Script 130
- DrRestore command, Conductor Script 135
- Dump/Restore facilities 41, 63, 126

E

- E1 governor state 106
- E2 governor state 106
- EMBRYONIC engine unit state 107
- engine
 - access, restricting 71
 - described 27
 - development engines 83
 - failover 33
 - functions 36
 - management tasks 37
 - management tools 40
 - monitoring with Conductor Script 239
 - monitoring with iIS Console 118
 - performance 89
 - performance tuning 116
 - process execution and 28
 - production engines 83

- engine (*continued*)
 - reconfiguration 113
 - recovery 34, 166
 - registration manager 142
 - relation to iIS process management system 45
 - relocating 62
 - shutting down 124
 - shutting down components 243
 - starting with Conductor Script 235
 - starting with iIS Console 110
 - startup, monitoring 112
 - states, changing with Conductor Script 242
 - states, changing with iIS Console 122
 - status, showing 309
- engine component
 - backup engine unit 33
 - compiled vs. interpreted 50
 - database service 32
 - engine database 32
 - engine unit 32
 - governor 34
 - icons, state indicators 69
 - monitoring with Conductor Script 240
 - monitoring with iIS Console 119
 - partitioning 85
 - partners 33
 - shutting down 124
- engine component log file 210
- engine configuration
 - about 85
 - component memory settings 89, 101
 - configuration file 90
 - deleting 104
 - duplicating 104
 - failover 33
 - failover and load balancing 35
 - minimal, no failover 31
 - multiple 29
 - properties 88, 91
 - recover cursor size 102
- engine database
 - See also* current state tables; history log tables; registration log tables
 - configuration file 90
 - configuration properties 89, 97
 - controlling growth 125
 - defined 32
 - Dump/Restore facilities 41
 - failure 125
 - logging settings 89, 98, 115
 - managing 124
 - recovery options 125
 - relation to iIS process management system 45
 - schema 331
- engine database tables
 - about 124
 - list, alphabetical 329
 - list, category 327
 - registration table entries 140
 - registration tables 346
- Engine Event Filter window 217
- engine instrumentation
 - performance chart 222
 - ShowStatus command 311
- engine management, Conductor Script
 - commands 264
- engine partitioning 86, 99
- engine server node
 - defined 52
 - installation 58
- engine sessions, *See* sessions
- engine state
 - changing 122
 - monitoring 117
- engine unit
 - configuring 99
 - defined 32
 - failover 33
 - failure 159
 - monitoring 120
 - startup 106
 - state, changing 122
 - states 107
- EngineUnitForceStandby alarm 205
- EngineUnitOnline alarm 205
- EngineUnitShutdown alarm 205
- environment mode, Conductor Script
 - commands 263

exception alarm 205
 exceptions, logged 210
 execution options 103
 Exit command, iIS Console 71
 EXPIRED timer state 199, 357

F

failover, engine unit 33
 failure
 engine 34, 159
 engine database 125
 network connection 159
 filtering
 engine events 217
 iIS Console lists 73
 FindDBService command, Conductor Script 283
 FindEngine command, Conductor Script 236, 239, 284
 FindGovernor command, Conductor Script 284
 FindNode command, Conductor Script 236, 285
 FindParentEngine command, Conductor Script 285
 FindPlan command, Conductor Script 244
 FindPrimary command, Conductor Script 285
 FindUnit command, Conductor Script 286
 first activity, description 165
 FlushLog command, Conductor Script 286

G

GenerateDistribution command, Conductor Script 292
 governor
 configuring 99
 defined 34
 monitoring 121
 partitioning guidelines 87
 startup 105
 states 106
 GovernorDisconnect alarm 205

H

history log schema, graphical representation 198
 history log tables
 database schema 347
 engine database 124, 327
 graphical representation 198

I

IDLE governor state 106
 IIOPServer command, Conductor Script 287
 iIS Console
 command summary 78
 engine list 69
 exiting 71
 list views 77
 lists, filtering 73
 lists, sorting 76
 main window 68
 menu bar 70
 online help 70
 overview 40, 65
 password protection 71
 popup menu 70
 starting 67
 window refresh behavior 72
 iIS distributions, registration 144
 iIS process engine, *See* engine
 iIS process management system
 See also system maintenance
 about 46
 components 45
 configuration 51
 installation program 56
 multiple engines 29
 resource considerations 54
 setup preparation 44
 setup procedure 55
 system software 47
 uninstalling 64
 implicit unregistration 142, 144

- installation
 - central server 57
 - development client node 59
 - engine server node 58
 - runtime client node 60
- instrument data, logged 211

L

- last activity, description 165
- library distribution and registration 139
- ListActivities command, Conductor Script 287
- ListActivityQueues command, Conductor Script 287
- ListConductorDistributions command, Conductor Script 288
- ListEngines command, Conductor Script 239, 240, 289
- ListFilters command, Conductor Script 289
- ListPlans command, Conductor Script 244
- ListProcesses command, Conductor Script 289
- ListRegistrations command, Conductor Script 290
- ListSessions command, Conductor Script 291
- ListTimers command, Conductor Script 291
- ListTransactions command, Conductor Script 291
- load balancing 35
- log file
 - about 210
 - engine component 212
 - registration 145
- logging settings, engine database 98

M

- Maximum number of processes option 103
- Maximum number of sessions option 103
- memory options, engine component 101
- memory requirements 116
- Memory-resident process limit 103

- message filters
 - iIS 214
 - specifying 212, 218
- messages
 - broadcasting 179
 - sending to sessions 177
- ModLogFlags command, Conductor Script 292
- monitoring
 - engine startup 112
 - engine state 118
 - process execution 179, 266
 - sessions 174
 - two-phase commit operations 258

N

- network failure 159

O

- OFF timer state 199, 357
- offered activities
 - described 165
 - execution of 168
- ON timer state 199, 357
- ONLINE engine unit state 107
- online help
 - iIS Console 70

P

- partitioning engine components 85
- partner 33
- password
 - Conductor Script 300
 - iIS Console 71
- PDF files, viewing and searching 24
- PENDING activity state 162, 199, 357

- performance chart 221, 224
- performance indicators 222
- performance, engine 89, 116
- ping interval, session 159
- PREPARE state (two-phase commit) 201
- primary engine unit, partitioning guidelines 86
- process attributes
 - locks, removing 194
 - value, changing 194
 - value/lock state, checking 189
- process client applications, *See* client applications
- process creation 166
- process definition, registration 145
- process development node 52
- process development workshops 45
- process engine, *See* engine
- process execution
 - about 165
 - activity creation 167
 - activity execution 168
 - activity termination 171
 - analysis 197
 - bottlenecks, checking 196
 - configuration options 89
 - historical data 197
 - managing 116, 179, 197, 266
 - monitoring 179, 266
 - monitoring and management tasks 180
 - objects in engine, diagram 179
 - process creation 166
 - process termination 173
 - recovering state 34, 123
 - recovery levels 123
 - reports 197
 - tuning 116
- process history 197
- process instance states
 - ABORTED 173, 198, 357
 - ACTIVE 198, 357
 - COMPLETED 173, 198, 357
 - CREATED 198, 357

- process instances
 - aborting 181
 - execution options 102
 - stalled 173
 - status, checking 181
 - swapping 116
- process, analysis of 197
- ProcessAbort alarm 205
- production engines 83
- production loads and memory 116
- properties
 - database configuration 97
 - engine configuration 88
 - performance chart 224

Q

- queued activities
 - described 165
 - execution of 169
- Quit command, iIS Console 71

R

- READY activity state 162, 199, 357
- ReadyActivity command, Conductor Script 293
- reconfiguring
 - engine 113
 - engine database logging 115
- reconnect parameters, sessions 160
- recover cursor size, process instances 102
- recovery
 - engine 34
 - engine database options 125
 - levels 123
 - state information 34, 123, 166
- refresh behavior 72

RegisterAlias command, Conductor Script 294

RegisterAssignmentRules command, Conductor Script 294

registering

- Conductor distributions 145
- engine with alarm service 206

RegisterProcessDefinition command, Conductor Script 295

RegisterUserProfile command, Conductor Script 295

RegisterValidation command, Conductor Script 296

registration

- about 137–142
- aliases 144, 148
- assignment rule dictionary 144
- engine registration manager 142
- entities registered 138
- iIS distributions 144, 243
- implicit unregistration 142, 144
- library distributions, and 139
- order 145
- performing 145
- process definition 145
- steps 139
- unregistration 142, 147
- viewing 150

registration log tables

- about 124
- entries, engine database 140
- list, alphabetical 346
- list, category 327

RemoveReadLock command, Conductor Script 297

RemoveWriteLock command, Conductor Script 297

reports, generating 197

repository

- central 38
- management tools 41
- setting up 54

repository server node 52

repository server, managing 62

requested message output, logged 210

restoring

- See also* recovery
- engine database tables 41
- session connections 159

RollbackActivity command, Conductor Script 298

RollbackTransaction command, Conductor Script 298

runtime client node

- defined 52
- installation 60

S

schema, *See* engine database

scripts

- comments in 232
- running using the `cscript -i` flag 230

SendMessage command, Conductor Script 299

session states

- ACTIVE 158, 199, 357
- SUSPENDED 158, 199, 357
- TERMINATED 199, 357

sessions

- about 158
- auto-reconnect 159
- disconnect parameters 160
- disruptions 159
- management 174
- messages, broadcasting to 179
- messages, sending to 177
- monitoring 174
- reconnect parameters 160
- setting maximum number 103
- suspending 161, 176
- terminating 161, 176

SetAttributeValue command, Conductor Script 300

SetPassword command, Conductor Script 300

SetPrimary command, Conductor Script 301

SetQueuedActivityPriority, Conductor Script 301

SetState command, Conductor Script 242, 302

SetTimer command, Conductor Script 302

SetTimerDeadline command, Conductor Script 303

SetTimerElapsed command, Conductor Script 304

ShowActivity command, Conductor Script 304

ShowActivityQueue command, Conductor Script 305

ShowConfiguration command, Conductor Script 306

ShowEngine command, Conductor Script 306
 ShowLogFlags command, Conductor Script 307
 ShowProcess command, Conductor Script 307
 ShowSession command, Conductor Script 308
 ShowStatus command, Conductor Script 239, 240, 309
 ShowTimer command, Conductor Script 313
 shutdown 124, 243
 Shutdown command, Conductor Script 243, 314
 sorting lists, iIS Console 76
 stall, process instance 173
 STANDBY engine unit state 107
 StartActivity command, Conductor Script 314
 StartDBService command, Conductor Script 315
 StartEngine command, Conductor Script 235, 315
 StartGovernor command, Conductor Script 316
 StartTimer command, Conductor Script 317
 StartUnit command, Conductor Script 317
 startup phases, primary engine unit 108
 StartupFailure alarm 205
 state

- database service 109
- engine unit 107
- process state values 198, 357
- process state, recovering 34
- recovering process execution state 123

 state transitions, activities 164
 StopTimer command, Conductor Script 319
 subprocess activities

- aliases and 143
- described 165
- execution of 170

 SuspendAllSessions command, Conductor Script 319
 SUSPENDED session state 158, 199, 357
 suspending sessions 161, 176
 SuspendSession command, Conductor Script 320
 swap-out interval 103, 116
 synchronizing transactions 200
 system design 51
 system libraries 47

system maintenance

- engine, relocating 62
- iIS system, uninstalling 64
- nodes, adding 61
- private repository, creating 63
- repository server 62

 system management node 52

T

TerminateAllSessions command, Conductor Script 320
 TERMINATED session state 199, 357
 TerminateSession command, Conductor Script 321
 terminating

- process instances 173
- sessions 161

 timers

- expiration time, changing 192
- state, changing 192
- states 199, 357
- status, checking 191

 Trace window

- about 212
- client messages, writing to 217
- message filters, setting 215
- using 215

 transactions, *See* two-phase commit
 troubleshooting

- engine event messages, displaying 217
- engine problems 204

 tuning process execution 116
 two-phase commit

- about 199
- managing transactions 202
- monitoring transactions 258
- PREPARE state 201

U

- UDS runtime environment [44](#)
- Uninstall command, Conductor Script [321](#)
- uninstalling iIS [64](#)
- UnRegisterAlias command, Conductor Script [322](#)
- UnRegisterAssignmentRules command, Conductor Script [322](#)
- UnRegisterProcessDefinition command, Conductor Script [323](#)
- UnRegisterUserProfile command, Conductor Script [324](#)
- unregistration
 - about [142](#)
 - aliases [150](#)
 - Conductor distributions [147](#)
 - iIS distributions [142](#)
 - implicit [142](#), [144](#)

- upgrading
 - Conductor system software [63](#)
 - monolithic [154](#)
 - rolling [155](#)
- user profile registration [144](#)
- UserAccess alarm [205](#)
- using list views, iIS Console [77](#)

V

- validation, registration [144](#)

W

- WaitForStartup command, Conductor Script [324](#)
- WFEEnvAgent, registration service [139](#)