

PeopleSoft®

---

EnterpriseOne 8.93  
Virtual AutoPilot  
PeopleBook

---

**May 2004**



EnterpriseOne 8.93  
Virtual AutoPilot PeopleBook  
SKU TT893VAP0504

Copyright© 2004 PeopleSoft, Inc. All rights reserved.

All material contained in this documentation is proprietary and confidential to PeopleSoft, Inc. ("PeopleSoft"), protected by copyright laws and subject to the nondisclosure provisions of the applicable PeopleSoft agreement. No part of this documentation may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, including, but not limited to, electronic, graphic, mechanical, photocopying, recording, or otherwise without the prior written permission of PeopleSoft.

This documentation is subject to change without notice, and PeopleSoft does not warrant that the material contained in this documentation is free of errors. Any errors found in this document should be reported to PeopleSoft in writing.

The copyrighted software that accompanies this document is licensed for use only in strict accordance with the applicable license agreement which should be read carefully as it governs the terms of use of the software and this document, including the disclosure thereof.

PeopleSoft, PeopleTools, PS/nVision, PeopleCode, PeopleBooks, PeopleTalk, and Vantive are registered trademarks, and Pure Internet Architecture, Intelligent Context Manager, and The Real-Time Enterprise are trademarks of PeopleSoft, Inc. All other company and product names may be trademarks of their respective owners. The information contained herein is subject to change without notice.

#### *Open Source Disclosure*

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>). Copyright (c) 1999-2000 The Apache Software Foundation. All rights reserved. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

PeopleSoft takes no responsibility for its use or distribution of any open source or shareware software or documentation and disclaims any and all liability or damages resulting from use of said software or documentation.



# Table of Contents

---

<b>About These EnterpriseOne PeopleBooks Preface</b>	<b>1</b>
EnterpriseOne Application Prerequisites .....	1
Obtaining Documentation Updates .....	2
Typographical Conventions and Visual Cues .....	2
Typographical Conventions.....	2
Visual Cues .....	3
Comments and Suggestions .....	3
<b>Data Capture for Virtual AutoPilot Scripts</b>	<b>4</b>
Data Capture Components .....	4
AutoPilot.....	5
PeopleSoft Software and AutoPilot Code .....	5
Event Stream.....	8
AutoPilot Playback Results Detail Table (F97214) .....	9
<b>EnterpriseOne Virtual AutoPilot Components</b>	<b>10</b>
Virtual Script Editor .....	11
Event Pane.....	12
Event Graph .....	14
Parameter Detail Pane.....	14
Script List Pane .....	15
Parameter Value Linking.....	16
Source and Target Parameter Identification .....	17
Looping Identification .....	18
hRequest Handle Value Linking.....	20
Thread Identification.....	20
Timing Interval Maintenance .....	21
Virtual AutoPilot Script Generation .....	22
Virtual Script Player.....	23
Virtual Script Player Initialization File Parameters .....	23
Virtual Script Player Command Line .....	26
Environment Initialization .....	27
Modes of Operation.....	27
Preprocessing of Valid Values List Data .....	27
Valid Values List Processing.....	27
Date Formatting.....	28
Script Failure .....	28
Virtual Script Player Limitations .....	28
VSMEditor .....	29
All Virtual Scripts List Box .....	29
Master Scripts List Box .....	29
VSM Files.....	30
Virtual Runner .....	30

Player Session Columns .....	30
Actions Tools .....	31
<b>Creating Virtual Scripts</b> .....	<b>32</b>
Capturing and Importing Test Results .....	32
Capturing Test Results .....	33
Importing Test Results .....	34
Viewing Test Results .....	34
Editing the Virtual Script .....	35
Using the Find Feature .....	36
Adding Loops .....	36
Value Linking Parameters .....	37
Linking Values in Inquiry Scripts .....	39
Linking Values in Entry Scripts .....	39
Generating the Virtual AutoPilot Script .....	40
Creating Master Scripts .....	41
<b>Running Virtual Scripts</b> .....	<b>42</b>
Running Virtual Scripts from a Single Workstation .....	42
Running Virtual Scripts from a Command Line .....	42
Running Virtual Scripts Using Virtual Runner .....	43
Launching and Managing Multiple Script Playback .....	45
Defining a Script .....	45
Defining the Host Machine .....	45
Defining Virtual Users .....	45
Gathering LoadRunner Results .....	46
Running Virtual Playback from the LoadRunner Controller .....	46
<b>Special Considerations for Simulated Playback</b> .....	<b>47</b>
Playback Timing .....	47
API Playback Timing .....	47
Interthread Timing .....	49
Call Level .....	51
Synchronous and Asynchronous Calls .....	52
Think Times .....	53
<b>Virtual AutoPilot Troubleshooting Tips and Techniques</b> .....	<b>54</b>
Locating the Causes of Virtual AutoPilot Script Failures .....	54
Finding Error Entries in the Virtual AutoPilot Log File .....	55
Locating the Log File in the Event of Early Script Failure .....	56
Setting the MessageLevel Parameter .....	56
Identifying an Environment Problem .....	57
Debugging Virtual AutoPilot Scripts .....	59

Displaying Business Function Parameters .....	59
Diagnosing Business Function Failures in PeopleSoft Explorer.....	60
Troubleshooting Value Linking Errors.....	61
Verifying the Validity of Virtual AutoPilot Script Data.....	64
Identifying and Correcting Duplicate Key Errors.....	64
Rectifying Irregular Transaction Times .....	65
Preventing Multiple Script Playback Problems .....	66
Correcting Uninitialized User Handle Errors .....	67
<b>EnterpriseOne PeopleBooks Glossary</b>	<b>68</b>
<b>Index</b>	<b>105</b>



# About These EnterpriseOne PeopleBooks Preface

EnterpriseOne PeopleBooks provide you with the information that you need to implement and use PeopleSoft EnterpriseOne applications.

This preface discusses:

- EnterpriseOne application prerequisites
- Obtaining documentation updates
- Typographical elements and visual cues
- Comments and suggestions

---

## Note

EnterpriseOne PeopleBooks document only fields that require additional explanation. If a field is not documented with the process or task in which it is used, then either it requires no additional explanation or it is documented with common elements for the section, chapter, PeopleBook, or product line.

---

---

## EnterpriseOne Application Prerequisites

To benefit fully from the information that is covered in these books, you should have a basic understanding of how to use EnterpriseOne applications.

See the *Foundation Guide*.

You might also want to complete at least one EnterpriseOne introductory training course.

You should be familiar with navigating the system and adding, updating, and deleting information by using EnterpriseOne menus and forms. You should also be comfortable using the World Wide Web and the Microsoft Windows or Windows NT graphical user interface.

These books do not review navigation and other basics. They present the information that you need to use the system and implement your EnterpriseOne applications most effectively.

---

## Obtaining Documentation Updates

You can find updates and additional documentation for this release, as well as previous releases, on the PeopleSoft Customer Connection Website. Through the Documentation section of PeopleSoft Customer Connection, you can download files to add to your PeopleBook Library. You can find a variety of useful and timely materials, including updates to the full PeopleSoft documentation that is delivered on your PeopleBooks CD-ROM.

---

### Note

Before you upgrade, you must check PeopleSoft Customer Connection for updates to the upgrade instructions. PeopleSoft continually posts updates as the upgrade process is refined.

---

### See Also

PeopleSoft Customer Connection Website, <http://www.peoplesoft.com/corp/en/login.jsp>

---

## Typographical Conventions and Visual Cues

This section discusses:

- Typographical conventions
- Visual cues

### Typographical Conventions

The following table contains the typographical conventions that are used in EnterpriseOne PeopleBooks:

Typographical Convention or Visual Cue	Description
<i>Italics</i>	Indicates emphasis, topic titles, and titles of PeopleSoft or other book-length publications. Also used in code to indicate variable values.
Key+Key	A plus sign (+) between keys means that you must hold down the first key while you press the second key. For example, Alt+W means hold down the Alt key while you press W.
Monospace font	Indicates a PeopleCode program or other code example.
“ ” (quotation marks)	Indicates an adjective that is used in a way that might not be readily understood without the quotation marks, for example "as of" date, "as if" currency, "from" date, and "thru" date.

Cross-references	EnterpriseOne PeopleBooks provide cross-references either below the heading “See Also” or preceded by the word See. Cross-references lead to other documentation that is pertinent to the immediately preceding documentation.
------------------	--

## Visual Cues

EnterpriseOne PeopleBooks contain the following visual cues:

- Notes
- Cautions

### Notes

Notes indicate information that you should pay particular attention to as you work with the PeopleSoft system.

---

#### Note

Example of a note.

---

### Cautions

Text that is preceded by *Caution* is crucial and includes information that concerns what you must do for the system to function properly.

---

#### Caution

Example of a caution.

---

---

## Comments and Suggestions

Your comments are important to us. We encourage you to tell us what you like, or what you would like to see changed about PeopleBooks and other PeopleSoft reference and training materials. Please send your suggestions to:

PeopleSoft Product Documentation Manager, PeopleSoft Inc., 4460 Hacienda Drive, Pleasanton CA 94588

Or you can send e-mail comments to [doc@peoplesoft.com](mailto:doc@peoplesoft.com).

While we cannot guarantee an answer to every e-mail message, we will pay careful attention to your comments and suggestions.

# Data Capture for Virtual AutoPilot Scripts

Several APIs enable PeopleSoft software to interact with any database or application server. The APIs communicate with PeopleSoft software's middleware, which serves as the conduit for runtime data flowing from the client workstation to the server and back again. PeopleSoft automated testing tools capture this data, which provides you with the raw material to build a virtual script that accurately simulates PeopleSoft software processes.

The following components of PeopleSoft automated testing tools architecture work together to capture, record, and store data about PeopleSoft software processes, including the parameters of all API calls and all other PeopleSoft software runtime events:

- AutoPilot, which allows you to write and play back a script to test PeopleSoft software applications and to configure script playback so that AutoPilot captures and saves playback data
- Hooks, or code, that reside in PeopleSoft software and in AutoPilot that capture and record data generated by the playback of an AutoPilot script
- Event stream, which is a time-stamped, chronological record of each AutoPilot and PeopleSoft software event that occurs during script playback
- AutoPilot Playback Results Detail Table (F97214), which stores the event stream

The placement of PeopleSoft software code is important to the creation of Virtual AutoPilot scripts. Because this code is positioned at the boundary between the PeopleSoft software runtime engine and the PeopleSoft software middleware, it captures data passing to the JDB and CallObject APIs before the APIs are routed to servers by the OCM. Therefore, you can reuse Virtual AutoPilot scripts regardless of changes to OCM mappings.

---

## Data Capture Components

Creating a virtual script requires that you first capture data from a PeopleSoft software session in which you launch an application, press buttons, enter data to header controls, and so on. The automated testing tool architecture of which AutoPilot is a part enables you to capture the events of a PeopleSoft software session by writing a script, configuring it for event capture, playing it back, and storing its results. You accomplish these tasks using the following three components:

<b>AutoPilot</b>	Allows you to write a script, play it back in PeopleSoft software, and save the results of the playback.
<b>Code that resides in PeopleSoft software and in AutoPilot</b>	Captures and records the data generated by AutoPilot and by PeopleSoft software during script playback.
<b>AutoPilot Playback Results Detail Table (F97214)</b>	Stores the data generated during script playback as an event stream, which is a continuous record of every PeopleSoft software and AutoPilot event that occurred during script playback.

# AutoPilot

The process of creating a virtual script begins with AutoPilot, which you use to write a script that tests PeopleSoft software processes. Playing back an AutoPilot script simulates PeopleSoft software activities, but only as initiated by one user. However, you can capture the results of script playback, including the processes generated, save the data, and use it to create a virtual script that you run to simulate more than one user.

## AutoPilot Script Creation

To begin the process of capturing data, you first write an AutoPilot script to test software processes such as launching applications, pressing buttons, entering data to header controls, and so on.

## AutoPilot Playback Configuration

You capture data about software processes by playing back the AutoPilot script, but you must configure playback for data capture. Your configuration choices establish how much data you capture and ensure that AutoPilot saves the data.

You can capture data at one of two levels:

- Level 1 captures data only for initiating API calls that run alone or call other APIs. If you choose this option, you capture data about only these APIs.
- All API calls capture data not only about level 1 API calls but about any API calls spawned by a level 1 API.

You also configure script playback to save and display results data after playback.

## AutoPilot Script Playback

After you have written a script and configured playback to capture the results, you play back the script. AutoPilot captures the playback data using internal code and code placed in the software.

# PeopleSoft Software and AutoPilot Code

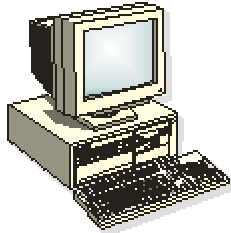
Code is strategically placed in AutoPilot and in 32 JDB functions and 1 CallObject function in PeopleSoft software to gather and store during AutoPilot script playback.

The placement of PeopleSoft software code provides the following advantages for creating EnterpriseOne Virtual AutoPilot scripts:

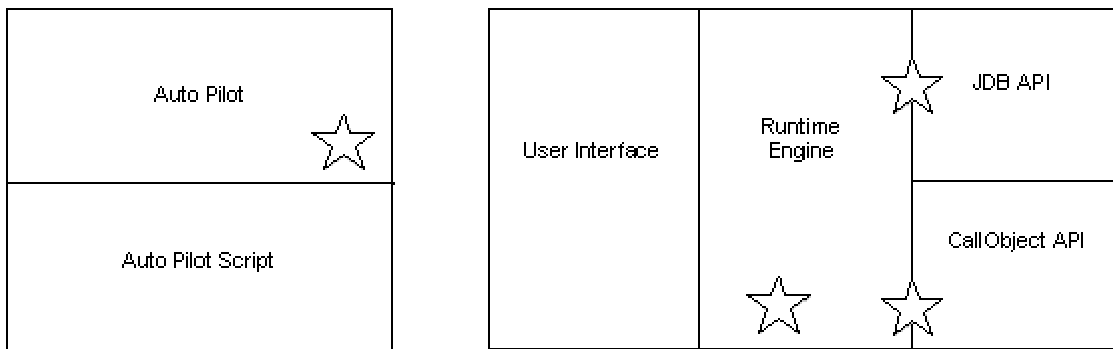
- Comprehensive data capture. Because code is positioned to capture both JDB and JDE CallObject API calls, you capture both database and business function activity.
- Simplified script maintenance. Because the code resides in slightly more than 30 JDB and CallObject functions combined, making changes in PeopleSoft software code is relatively easy.
- Flexibility in running scripts. Data that you capture can be run independent of platform or Object Configuration Manager (OCM) mapping considerations.

The stars in the following diagram illustrate the placement of PeopleSoft software and AutoPilot code for the capture of data during playback of an AutoPilot script:

### Code Placement for Scripting Tool



Workstation



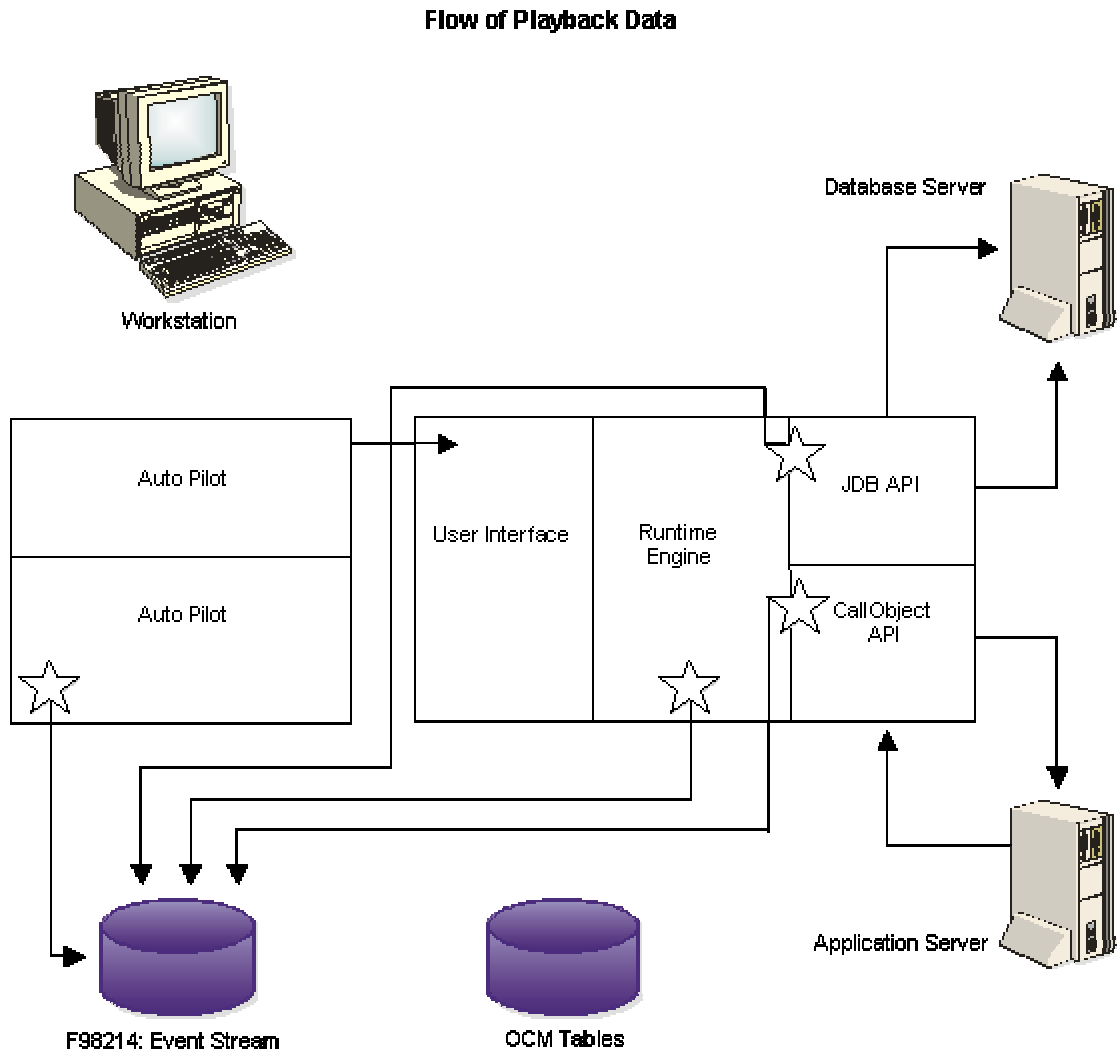
Code placed in PeopleSoft software performs the following functions that lay the groundwork for the creation of a Virtual AutoPilot script:

- Captures parameter data on JDB and CallObject API calls that occur during the playback of an AutoPilot script
- Writes the parameter data to a file-mapping object that PeopleSoft software shares with AutoPilot
- Writes data on event rules, button presses, and event timing to the file-mapping object

Code placed in AutoPilot performs the following functions that lay the groundwork for the creation of a Virtual AutoPilot script:

- Writes data on AutoPilot events to the file-mapping object
- Copies the PeopleSoft software and AutoPilot data in the shared file space into a BLOB (Binary Large Object) field in the AutoPilot Playback Results Detail Table (F97214)
- Enables the AutoPilot user to access the AutoPilot Playback Results Detail Table (F97214)

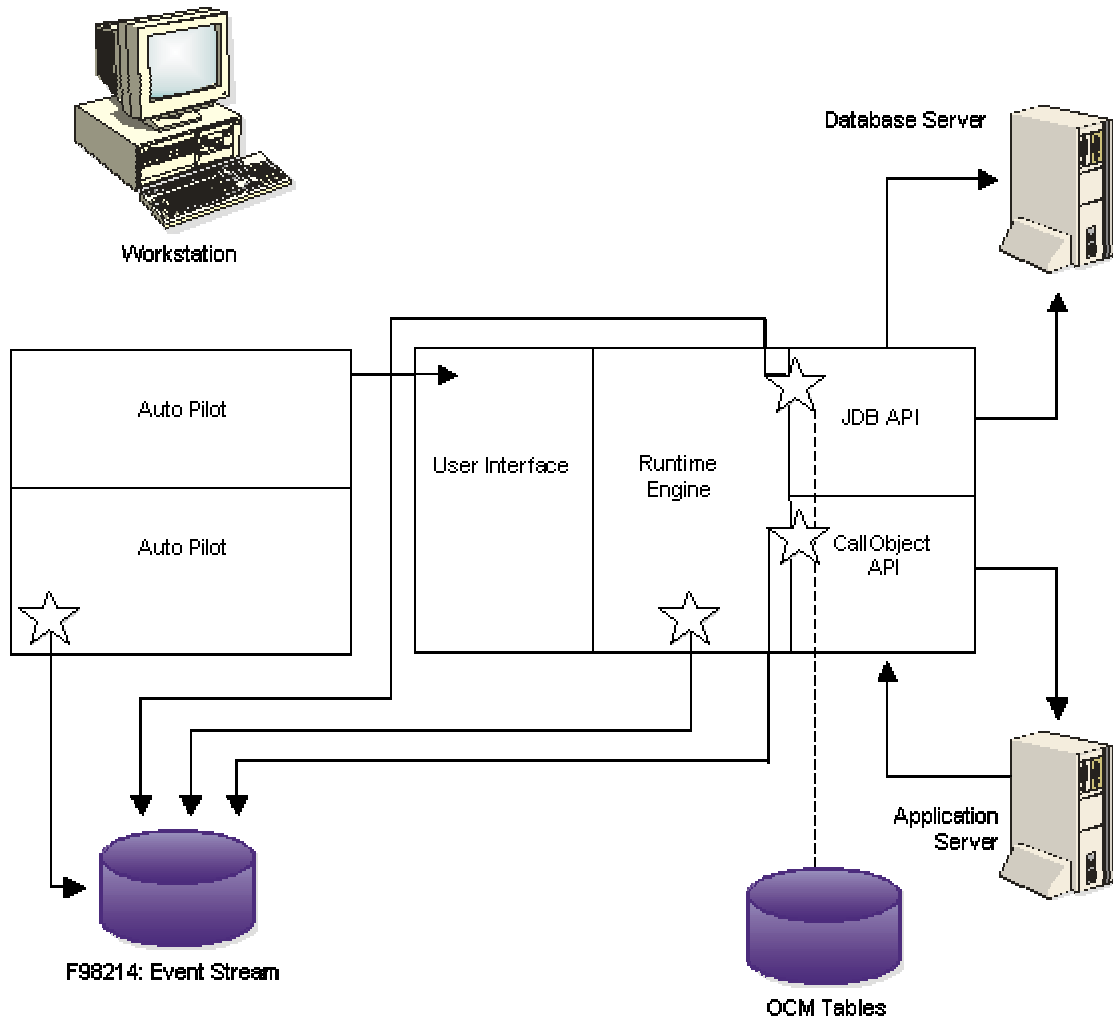
The following diagram illustrates the flow of playback data from the data-capture points to the AutoPilot Playback Results Detail Table (F97214), where the results are stored:



Virtual AutoPilot scripts that you create using the data captured from an AutoPilot script are platform independent and can be run on any operating system with any OCM mappings because PeopleSoft software code captures API call data before it reaches the OCM for mapping.

The dotted line in the following diagram illustrates the positioning of PeopleSoft software data capture code in relation to the OCM.

## Flow of Playback Data in Relation to the OCM



## Event Stream

You generate an event stream when you run an AutoPilot script that you have configured to capture playback results. The event stream is a time-stamped, chronological record of every AutoPilot and PeopleSoft software event that occurs during playback, including:

- JDB and CallObject API calls
- Thread identification
- Event rules
- PeopleSoft software error and warning messages
- AutoPilot events confirming that the script and PeopleSoft software are on the same form

## AutoPilot Playback Results Detail Table (F97214)

The AutoPilot Playback Results Detail Table (F97214) contains the results of all AutoPilot playback sessions that you have captured and saved. You can access the event stream for a script playback session through AutoPilot, the Virtual Script Editor, or EnterpriseOne Analyzer Tool.

Using AutoPilot, you can view a summary of every event that occurred during script playback. For example, you can view the following information for any test on the test Results form:

- Results sets
- Summary (Script, Release, Machine, and so on)
- JDE.INI and JDE and JDEDEBUG.LOG
- Screen captures
- Messages
- Results

From the Virtual Script Editor or from EnterpriseOne Analyzer Tool, you can view each event in more detail. For example, you can choose an API call and view the input and output parameter values for the call. If you import an event stream record from the database table to the Virtual Script Editor, you can modify the record so that you can play it as a Virtual AutoPilot script.

# EnterpriseOne Virtual AutoPilot Components

You use components that are external to EnterpriseOne Virtual AutoPilot (Virtual AutoPilot) to capture, record, and store data generated by an AutoPilot playback session. These components include the following:

- AutoPilot
- Data-capture code in AutoPilot and in PeopleSoft software
- Event stream
- AutoPilot Playback Results Detail Table (F97214), where the event stream from the playback session is stored

The internal components of Virtual AutoPilot allow you to complete the Virtual AutoPilot scripting process. These components are:

- Virtual Script Editor
- Virtual Script Player
- VSMEditor
- Virtual Runner

You use the internal components of Virtual User Tool to complete the following tasks:

- Import an event stream into the Virtual Script Editor
- Modify the event stream by adding rules that govern the passing of parameters and looping (repeated JDB Fetch calls to complete a database inquiry)
- Use the Virtual Script Editor to automatically add rules to handle thread identification and hRequest handles
- Generate a Virtual AutoPilot script
- Run a Virtual AutoPilot script on the Virtual Script Player
- Use the VSMEditor to concatenate a series of individual Virtual AutoPilot scripts into one master script
- Use Virtual Runner to manage script playback, either from a single workstation or from multiple workstations

You also can manage script playback using LoadRunner from Mercury Interactive.

---

# Virtual Script Editor

The Virtual Script Editor allows you to create and generate a PeopleSoft Virtual AutoPilot (Virtual AutoPilot) script that you can use to simulate the activity of many concurrent users. Working with the Virtual Script Editor represents the second step in a three-step process of producing a Virtual AutoPilot script playback session. You create a Virtual AutoPilot script playback session by completing the following steps in the following order:

1. Capture data generated by AutoPilot script playback and store the event stream in a results repository.
2. Use the Virtual Script Editor to modify an event stream and to generate a Virtual AutoPilot script that contains all the information required by the Virtual Script Player to simulate the activities of the system's runtime engine.
3. Play back a modified event stream (the Virtual AutoPilot script) using the Virtual Script Player.

The event stream is a chronological, time-stamped record of every event that occurs during the playback of an AutoPilot script, including the following:

- User input
- Processing performed by the run-time engine, such as thread creation
- Event rules; informative messages
- API calls to the PeopleSoft middleware

AutoPilot performs no editing during the process. The event stream represents a record of the events that have already occurred. You cannot edit it by adding, deleting, or reordering data. To change it, you must generate a new one by modifying an existing AutoPilot script, or by creating a new script and then replaying it.

Using the Virtual Script Editor, you can do the following:

- View the titles of all the scripts whose results you stored in the AutoPilot Playback Results Detail Table (F97214)
- Import an event stream
- View an event stream as a single, continuous record
- View the timing of events by category, represented in a horizontal bar graph
- Choose an individual API call and view the input values sent to the server and the output values returned to the client workstation
- Create links between parameters of API calls so that parameter values can be passed between calls during virtual script playback
- Identify and designate loops so that the virtual script can handle repetitive processing tasks, such as database retrieval

The Virtual Script Editor helps you address problems that you encounter when trying to create a script that you can run to simulate activities in a dynamic client/server system. Problems that you might encounter include the following:

- Identifying API parameters that require dynamic values
- Providing a way to pass values dynamically between API call parameters to avoid data conflict and record contention
- Making the values of hRequest handle parameter values dynamic to simulate concurrent user activity
- Synchronizing timing between events during script playback to keep processing running regardless of network stress placed on the server
- Synchronizing timing between data-dependent APIs in threads running asynchronously to avoid one API starting before another has finished processing
- Identifying repetitive processing tasks, such as database inquiry, so that the Virtual Script Player can efficiently simulate the work of the software

The Virtual Script Editor handles the following virtual script creation tasks automatically:

- Linking values of parameters in separate API calls so that values can be passed, provided that the calls meet certain criteria
- Storing the values of hRequest handles as variables
- Storing identification of thread IDs
- Storing information about time gaps between events in a single thread and between interthread-dependent events

You perform the following virtual script creation tasks manually:

- Linking values of parameters in separate API calls that do not meet all the criteria required for automatic value linking
- Identifying repetitive processes, such as database inquiry, as loops

After you have completed these manual tasks, you use the Virtual Script Editor to generate the virtual script. The Virtual Script Player receives from the Virtual Script Editor all of the information necessary to run the virtual script.

## Event Pane

You click the Import button on the Virtual Script Editor form to import the results from an AutoPilot script playback session. The Virtual Script Editor populates the event pane with the event stream. The event stream contains a time stamping of each event. Therefore, you can review AutoPilot events or API calls during playback that might have taken an unusual amount of time to run.

You use the event pane to view data about the following kind of playback events:

- CallObject APIs
- JDB APIs
- AutoPilot events
- Event rules

- Informative messages, including system errors and warnings
- Thread creation

The event pane also contains the following columnar information about each event:

- Timing information, such as the start, end, and elapsed time of an event
- Thread identification
- hUser handle identification
- hRequest handle identification
- Call level
- Message entry identifying the event
- Message information about an event, such as JDB call to open a table from memory cache

The message entry for each event includes an abbreviation that identifies the type of event that occurred. The following table summarizes the abbreviations and the type of event that each represents:

<b>Abbreviation in Message Column of Event Stream</b>	<b>Type of Script Event</b>
JDB	Database API call
RTE	CallObject API call
EVR	Event rule
LOG	System warning message
ERR	System error message
MSG	AutoPilot message
AUT	Action in AutoPilot (for example, typing to control)
THR	Thread action

You use the following buttons to change the view in the event pane:

- Generate** Allows you to generate a Virtual AutoPilot script. Click this button only after you have finished editing the event stream in the Virtual Script Editor.
- Filter** Allows you to remove unwanted events from the list by applying criteria found in the Filter form.
- View Log** Allows you to look at the log produced when you generate the Virtual AutoPilot script. The log includes the number of lines in the script and the number of errors, if any.

## Event Graph

You can view playback events by category in a horizontal bar graph by choosing the Graph option in the Virtual Script Editor. While the event stream pane presents the events of an AutoPilot script playback vertically, in a single chronological stream, the event graph presents the events horizontally across a timeline.

You can break up the chronology by message type, such as JDB API calls or event rules.

The event graph provides you with another detailed snapshot of activity that occurred during AutoPilot script playback. You can focus on events of unusual duration, which can be helpful in debugging applications, analyzing network activity, or rewriting and rerunning the original AutoPilot script.

## Parameter Detail Pane

You can view the parameters that make up an API call by clicking an API call event line in the event pane. The pane that appears shows the name of each parameter in the call and its value, if any. For example, the detail pane might display the value of the user handle parameter that a JDB call passes to the database.

This detail pane provides a complete snapshot of each API call at a given point in time. For example, the pane shows arrows that indicate the flow of data that occurred during the call. An arrow on the left side of the box next to the name of a parameter indicates that the call passed the value from the client workstation to the JDENET or database driver. An arrow on the right side of the box indicates that the call returned data from the server. In some cases, a box contains both arrows, indicating that data flowed in both directions.

The parameter detail pane offers a before-and-after snapshot of script playback. Before playback, parameters for a CallObject API, such as BatchOpenOnInitialization, contain no batch number or batch date parameter values. After playback, these parameters contain returned values.

The parameter detail pane also displays the parameters of API calls that pass an environment handle to the database.

Finally, many API calls contain a request handle that points to a particular place in memory that the run-time engine has allocated for the call. The parameter for the request handle appears in the parameter detail pane if the API call used a request handle.

The ability of the AutoPilot and PeopleSoft software hooks to capture data at this level of detail is critically important to Virtual AutoPilot because the goal of Virtual AutoPilot is to simulate, as closely as possible, the actual activities of the system. If the Virtual AutoPilot script does not have the complete parameters of an API call, it cannot accurately model the activities of the system and its interaction with the client workstation, the database server, and the application server.

## Script List Pane

The script list pane on the Virtual Script Editor form displays in chronological order the AutoPilot script playback results that you saved.

The script list pane displays script result information in the following columns:

<b>Column</b>	<b>Description</b>
Test	Database ID number assigned to each AutoPilot script playback session
Client	ID of the workstation on which you ran the test
Start Time	Date and time at which you ran the test
Elapsed	Time it took the test to run to a successful conclusion, failure, or cancellation
Environment	System environment against which you ran the test
Release	PeopleSoft software release against which you ran the test
Script	Name that you assigned to the test
Status	Result of the test—success, failure, or cancellation

After you choose a script, you choose one of the following buttons to manipulate the form view:

<b>Virtual Script Editor Form Button</b>	<b>Purpose</b>
Filter	Allows you to remove AutoPilot script playback results that you do not want, using criteria on the Filter form.
Import	Imports into the Virtual Script Editor the event stream from a test result that you choose.
Refresh	Refreshes the script list pane from the database.
Delete	Removes one or more tests from the database.

### See Also

- *Call Level* in the *Virtual AutoPilot Guide*

## Parameter Value Linking

After you import an event stream into the Virtual Script Editor, you are ready to create the virtual script. Using the Target Parameters and Source Parameters panes, you complete the task of value linking. Value linking ensures that the virtual script can pass parameter values from one API to another. You identify a value-containing parameter in a source parameter API call and link the value to a target parameter in another API call. This process ensures the passing of a parameter value from one API to another API that requires the value.

In addition, the values contained in many API call parameters must be dynamic. For example, each time a user performs voucher entry, the PeopleSoft software creates a new batch number, a function that is essential to prevent the creation of duplicate keys. Value linking ensures that the Virtual Script Player can simulate this function. When you link the parameter value of two API calls, the Virtual Script Editor stores the value as a variable, and the value changes each time you run a virtual script.

For example, a script might call the business function `BatchOpenOnInitialization`. For the parameter `ICU`, which is the batch number, suppose the API returns the value 5056. In turn, the script might call the business function `BeginDoc`, which uses the value 5056 as an input to the `ICU` parameter. To simulate multiple script playback, the value 5056 must change in order to reflect the new batch numbers returned each time people using the system make these API calls. As long as you have linked the parameters, the batch number parameter value will change each time you run a virtual script.

Value linking simulates the application logic that is used to run PeopleSoft software operations. It codifies the relationship between one API call and another. When you run the virtual script, the Virtual Script Editor passes to the Virtual Script Player the ID number of the source parameter that you link to the target parameter. The Virtual Script Player uses this information to pass parameter values between API calls.

Several types of data necessary to run a virtual script are candidates for value linking:

- The client host name, which could change any time a script is played back.
- Next numbers, which must change each time a script is run in order to avoid producing duplicate data that would break the script.
- Valid values lists used in AutoPilot scripts, which must be designated as such in a Virtual AutoPilot script so that, during run time, the Virtual Script Player draws new values from the list rather than using the same value repeatedly.

## Source and Target Parameter Identification

The Virtual Script Editor provides detailed information about API calls in the event stream when you click the Link Parameters button in the toolbar. The Virtual Script Editor identifies the API calls made during script playback as source parameters or target parameters. A source parameter contains a value that the system passes to a parameter in another API call. The parameter receiving the value is the target parameter.

Information about the source and target parameters appears in separate panes of the Virtual Script Editor form. Each of the panes contains the following information about API calls made during script playback:

<b>Column Heading in Target Parameters Pane and in Source Parameters Pane</b>	<b>Information Displayed</b>
ID	Displays in the source parameter pane a value that identifies that parameter. If you value-link the source parameter to a target parameter, the source parameter ID value appears next to the target parameter, along with a chain-link symbol indicating that you have linked the parameters.
Start Time	Specifies the time the event occurred during playback.
Thread	Identifies the thread generated by the system's run-time engine in which the event occurred.
Label	Identifies the data dictionary alias of the parameter.
Value	Shows the value of the parameter contained within the JDB or CallObject API call.
Comment	Contains the variable name of a business function parameter and the type of data that it contains.
Event	Identifies the specific JDB or CallObject API called or AutoPilot event in which a value was entered.

To see the complete set of parameters for an API call that occurred during AutoPilot playback, click an item in either pane. The Virtual Script Editor displays the parameter names and values for the selected call. Arrows indicate the direction of the flow of data.

The detail panes provide a snapshot of the API calls that the applications generated during AutoPilot playback. You can examine the parameter values and the flow of the data to help determine, for example, a parameter value used in one API call that the system passed to another API call later in the script.

To find the parameter of an API call in an event line, you might have to click a node in the detail pane to expand a tree. For example, for a JDB call, find the Value node in the detail pane and expand the tree to expose all of the column parameters in the database table. You then can search the column parameters for the source parameter for which you are looking.

## Automatic Value Linking of API Call Parameters

When you click the Link Parameters button on the Virtual Script Editor, the tool automatically links some source parameters of API calls to target parameters of other API calls. The Virtual Script Editor accomplishes this automatic linking according to a set of rules. The following rules govern the automatic linking of API calls:

- Data must have been entered in AutoPilot.
- The value of the target parameter must exactly match the value of the source parameter.
- The data dictionary ID of the target parameter must exactly match the data dictionary ID of the source parameter.

The Virtual Script Editor finds those parameters that meet each of these conditions and automatically links them.

## Looping Identification

Virtual AutoPilot requires a method to handle repetitive processing, such as that which occurs when you click the Find button on a Find/Browse form to perform a database inquiry. In this situation, a JDB Fetch call might return any number of values from the database. Looping rules provide a way to identify these repetitive retrievals. Having identified all of them in a single step, you can more easily link the values of source and target parameters.

You can specify the number of times you want the API to return to the database to retrieve values to be used as parameters in the script. This capability allows you to more accurately simulate the load placed on the system. You can increase or decrease this number, but the actual number of matches that the API returns, based on the inquiry command that you write in the AutoPilot script, will likely determine the number of loops that you specify.

## Manual Value Linking of API Call Parameters

If the source and target parameter do not meet all three of the following you must link them manually by clicking a parameter in each pane, and then clicking the Link Parameters button in the Virtual Script Editor tool bar.

- Source parameter must come from an Autopilot event (type to grid cell, for example)
- Data dictionary items must (called *label* in Virtual Script Editor) match in source and target
- Values must match in source and target

In deciding the target parameter value to link to a source parameter value, you:

- Match data dictionary aliases
- Match parameter values
- Choose, in general, the event in the Source Parameter pane whose start time most closely matches the start time of the event in the Target Parameters pane

You do not have to link the values of source and target parameters when:

- APIs do not contain data dictionary items

- An API call returns a zero or null value for the source parameter that might be value-linked to a target
- The data flow of the source parameter is indicated as bi-directional, but the input value and the return values are the same

You code as literal the values of any parameters that meet at least one of these criteria by clicking the Mark Literal button on the Virtual Script Editor form.

The content of the AutoPilot script also plays an important role in your decisions on value linking. If the AutoPilot script that you write contains a literal value that the script writes to a grid column or header control, you cannot make that literal value dynamic by linking. The Virtual Script Player will be forced to use that literal value repeatedly during Virtual AutoPilot script playback.

Because you cannot make literal values dynamic, avoid using them often in a Virtual AutoPilot script. The entry of the same value to a grid column or header control by multiple users does not accurately simulate the way people use the system. To set up a more realistic scenario when you write the AutoPilot script, create valid values lists containing more than one value. During Virtual AutoPilot script playback, the Virtual Script Player goes to the `.atd` directory on your hard drive to retrieve the list's values, and then it cycles through them, entering a different value in each simulated playback session until it reaches the end of the list, when it returns to the top of the list and repeats the cycle.

## Dynamic Loop Creation

You create a dynamic loop in the AutoPilot script by writing a command to press the Find button on a Find/Browse form. This command triggers a string of JDB API calls, culminating in a Fetch call.

AutoPilot and Poeplesoft software code records all of these events during playback; however, thousands of events might exist in an AutoPilot script. The Virtual Script Editor offers an easy way to locate the AutoPilot press Find button event and the repetitive processing that occurred because of the event. When you type the word Find in the locator space in the Virtual Script Editor form, the Virtual Script Editor highlights the first line in the event stream pane that contains the word Find. Then, you can scroll down the pane to the Fetch call.

Following the Fetch call, you can scroll through the event stream to locate the series of calls that resulted from the Fetch. This series of calls constitutes the loop.

## Dynamic Loop Designation

The Virtual Script Editor allows you to designate dynamic loops in the Virtual AutoPilot script. By doing so, you add looping rules to the script. These rules allow the Virtual Script Player to perform the repetitive processing that the system performs.

When you right-click the Fetch line in the event stream and choose Add Loop, the Virtual Script Editor produces a Dynamic Loop Manager form, which you use to apply a dynamic loop and to establish the rules by which the Virtual Script Player manages the loop at virtual run time.

You can instruct the Virtual Script Player to run the inquiry loop until the data runs out, or you can instruct it to loop a specific number of times.

When you click the Apply Loop button in the Dynamic Loop Manager, you establish the looping rule. The Virtual Script Editor indicates the loop in the event stream, in the Source Parameters pane, and in the Target Parameters pane by graying the sequence of API calls that are part of the loop.

## Dynamic Loop Editing

You can edit established looping rules before generating a Virtual AutoPilot script. You also can undo the loop if, for example, a series of calls does not constitute a loop. The ability to edit dynamic loops provides an added measure of control over the creation of Virtual AutoPilot scripts.

To edit the loop, you right-click the Fetch command line in the event pane and choose Edit Loop Details. On the Dynamic Loop Manager form, you can change the number of times you want the script to loop, or you can choose the Undo Loop option to remove the loop.

## hRequest Handle Value Linking

The Virtual Script Editor automatically stores hRequest handle parameter values for JDB API calls. This value represents the address of a memory block that the system allocates for storing information about an open table. The address provides you with entry to the database each time you need to open a table to perform a Fetch, FetchKeyed, SelectKeyed, FetchMatchingKey, or CloseTable function. However, when you create a Virtual AutoPilot script and play it back, the hRequest handle parameter value probably changes. Playback could not continue if this value were constant.

The Virtual Script Editor handles the problem by storing the hRequest handle parameter value as a variable and passing the variable to the Virtual Script Player during playback. The value of the hRequest handle variable changes to reflect the new address of a database table opened during script playback.

You can view the hRequest handle returned from the original API database call by clicking a call in the Source Parameters pane and viewing the details of the call. The Virtual Script Editor displays in the detail pane the request handle returned from the OpenTable API call.

If a database API call, such as OpenTable, leads to additional API calls, such as FetchKeyed and CloseTable, the Virtual Script Player passes the new memory address of the opened table to these subsequent calls. During virtual playback, the subsequent APIs use the new handle to run SQL statements and to close the table.

## Thread Identification

The Virtual Script Editor also stores the idThread numbers that AutoPilot gathers into the event stream during script playback. These identifier numbers represent the synchronous and asynchronous threads generated by the runtime engine. The runtime engine assigns each event to a thread and tags each thread with a number.

During virtual script playback, the Virtual Script Editor passes idThread parameters to the Virtual Script Player, which assigns different idThreads to each event and associates each script event with its new identifier.

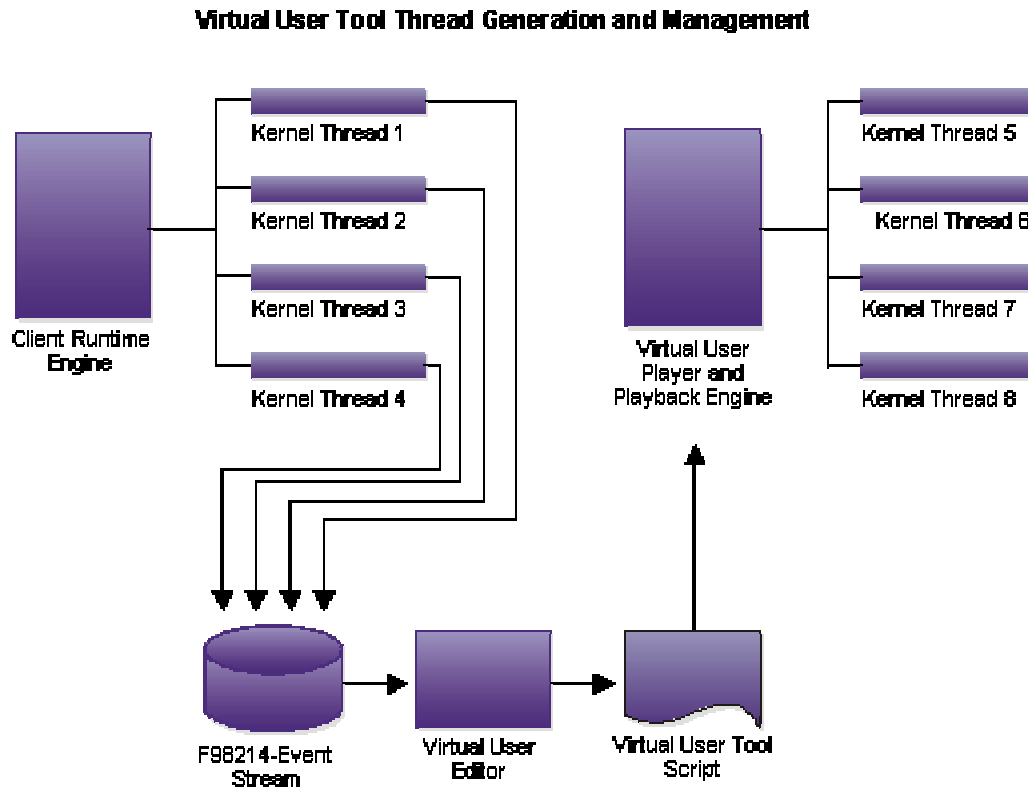
The following diagram illustrates a simplified view of EnterpriseOne Virtual User Tool's thread management strategy.

---

## Note

During Virtual User Tool script playback, the Virtual Script Player rennumbers the original threads generated during AutoPilot script playback. The Virtual Script Editor's role is to store the thread identification information and to pass it on through the virtual script.

---



## Timing Interval Maintenance

The Virtual Script Editor also automatically handles problems of timing that might emerge in the creation of a Virtual User Tool script. The time-stamped event stream log of events captures the length of time elapsed between each event. However, after you create a Virtual User Tool script, you do not know the different scenarios in which the Virtual User Tool script runs. For example:

- The workstation on which the script runs might be simulating 50 users.
- The power of the workstation might differ from the one on which the original script data was captured.
- The server against which the Virtual User Tool script runs might be more or less powerful than the server against which the original script ran.

These factors combine to make it likely that the time required by a Virtual User Tool script to run will differ from the time that the original script required to run.

The Virtual Script Editor handles this problem by preserving in the virtual script the time intervals that existed between events when you ran the original script. The time intervals represent the length of time required to carry out the processing between events. Thus, even if an API call during virtual script playback takes longer to carry out than the API call in the original script, the Virtual Script Player preserves the original time difference between one API call and the next.

The Virtual Script Player initialization file also contains timing parameters that govern the playback of the Virtual User Tool script. You can adjust, to a limited extent, some of these parameters; for example, you can adjust how fast the Virtual User Tool script plays back.

### See Also

The following topics in the *Virtual AutoPilot Guide*:

- *Virtual Script Player Initialization File Parameters* in the *Virtual AutoPilot Guide*
- *Playback Timing* in the *Virtual AutoPilot Guide*

## Virtual AutoPilot Script Generation

The Virtual AutoPilot script is the output from the Virtual Script Editor and the input to the Virtual Script Player. Virtual AutoPilot scripts appear in text file form with a header and the edited list of events that you captured during script playback, imported into the Virtual Script Editor, and edited, both manually and automatically.

For ease and consistency of interpretation, each event in the script is structured in a particular way. For example, each event begins with the letter *e* and is followed by a unique identifying number. In addition, each script identifies the environment and the network user, and contains an open table handle. However, it is not necessary that you look at a Virtual AutoPilot script in order to run it.

Virtual AutoPilot classifies the following three types of events and identifies them as such in the script:

<b>Functions</b>	Includes JDB and CallObject APIs
<b>Assignment statements</b>	Refers to values typed in AutoPilot
<b>Conditional</b>	Tests/branches (if/then statements)

Virtual AutoPilot divides each event into parts and, in turn, identifies each of the parts based on an assigned format and a unique value. In short, the Virtual AutoPilot script contains the details necessary for the Virtual Script Player to simulate running the PeopleSoft software kernel.

Virtual AutoPilot identifies transaction boundaries, which you can set in the original script by designating a script command as the start of the transaction and another script command as the end of the transaction. Setting transaction boundaries can help you to analyze system performance when running a series of tasks.

---

# Virtual Script Player

The Virtual Script Player uses the Virtual AutoPilot script that you generate in the Virtual Script Editor to simulate the concurrent activities of one or more PeopleSoft software users. It bypasses the presentation layer of PeopleSoft software and reproduces the PeopleSoft application calls to the JDB and CallObject middleware. This reproduction is based on the timing and the sequencing of data in the event stream that you generate with AutoPilot, manipulate in the Virtual Script Editor, and generate in modified form in the Virtual AutoPilot script. In essence, the Virtual Script Player assumes the role of the PeopleSoft software runtime engine.

## Virtual Script Player Initialization File Parameters

The `vap.ini` file is a text file that contains the parameters that define the way that the Virtual Script Player runs. These parameters govern the paths that the Virtual Script Player follows to find files, synchronize playback timing, and set playback speed.

You can change the parameters, within established limits, to set the way the Virtual AutoPilot scripts play.

### Command

The Command section of the `vap.ini` file contains the parameters that are necessary for interaction between Virtual Runner, which manages script playback, and the Virtual Script Player, which runs playback. These parameters specify the following:

- User ID and password
- Environment
- Script name
- Log file of summary playback statistics
- Location of the Virtual Script Player executable

The following table summarizes the [COMMAND] parameters and the meaning of each one:

Parameter	Meaning
UserID=	Virtual AutoPilot user ID. Override on command line by entering -u and a user ID.
Password=	Password for Virtual AutoPilot user. Override on command line by entering -p and a password.
Environment=	Environment for Virtual AutoPilot script playback. Override on command line by entering -e and an environment.
Script=playscript .vsx	Name of Virtual AutoPilot script (user can specify full path name for script here).

Common log= Log file to which Virtual Script Player will write summary statistics for all playback sessions. Default folder is vap\_logs. Used only with Virtual Runner.

Binname=d:\b7\system\bin32\vapplayer.exe Path by which Virtual Runner finds the Virtual Script Player executable.

## Paths

The Path section of the vap.ini file identifies the directories for files that are needed by the Virtual Script Player. The contents of the needed files are:

- Log file, which gives detailed information about each EnterpriseOne Virtual AutoPilot script playback session, the script name, and a line-by-line summary of each event in the script. The Virtual Script Player logs each event as it completes. The file also includes the start time and the date of the log.
- EnterpriseOne Virtual AutoPilot script file, which stores all scripts that you might use for virtual playback.
- Valid values list file, which stores any valid values lists that the Virtual Script Player draws on for input values to run business functions. The Virtual Script Player uses valid values lists to get a new value each time it runs a business function.

The default file paths are as follows:

File/Contents	Parameter in vap.ini file	Default Path
Log of EnterpriseOne Virtual AutoPilot playback events and messages	LogDirectory	c:\autopilot\VAP_LOGS
EnterpriseOne Virtual AutoPilot scripts	VirtualScripts	c:\autopilot\VSX
Valid value lists	ValidValueLists	c:\autopilot\ATD

## Timing

The Timing initialization parameters of the vap.ini file help you specify the terms under which Virtual AutoPilot scripts play back:

- Rendezvous of multiple playback sessions, to control the amount of time the Virtual Script player delays a playback session following a rendezvous of multiple scripts running on a single workstation
- Synchronization of playback events, to set limits on the amount of time that threads can be inactive, events can occur behind the start time scheduled by the script, or that a thread has to wait for an API value or a handle parameter
- Playback speed, to adjust the amount of time between events to compensate for a fast or slow client workstation

The following table lists the Virtual AutoPilot timing initialization parameters, their default values, what they govern, and the kind of timing factor to which they relate:

Parameter Name	Default Value	Meaning	Timing Factor
RandomDelayMax	0 seconds; can be set as high as 3,600	Allows user to set a maximum period that the Virtual Script Player will wait after the LoadRunner OWLogin rendezvous and environment initialization to begin each playback session. The default value means that following rendezvous, each player session proceeds without delay.	Rendezvous of multiple playback sessions
lMaxSleep	10,000 milliseconds	Establishes an upper limit on thread sleep time. Inactive threads must check on system status at least this often. If errors require the Virtual Script Player to shut down all threads, the parameter also determines the maximum amount of time required for the Player to shut down.	Playback synchronization
lTooLate	200 milliseconds; set higher in for debugging	The latest that any event can be run after the script schedules its start without causing virtual script playback to terminate.	Playback synchronization
lTimeout	60 seconds	Maximum number of seconds that an event has to run. If that number exceeds the parameter, Virtual Script Player terminates the playback session.	Playback synchronization
ClientSpeedFactor	100	Controls timing between script events by a constant factor. Decreasing the value of the parameter decreases the time between events.	Playback speed

## Log

You use the Log section of the `vap.ini` file to specify the type of messages that the Virtual Script Player writes to a log file during a Virtual AutoPilot script playback session. These messages can be important for debugging purposes. The following table summarizes the available log parameters and the debug message level that each one represents:

Log Parameter	Debug Message Level
31	Maximum log output; flush log file after each message (LoadRunner excluded)
15	Parameter values and value substitutions
7	Error, warning, and status messages

3	Error and warning messages
1	Error messages only
0	Minimal messages

---

**Note**

You can cause the log file buffer to flush after every message by adding 16 to any parameter less than 31. However, you should not routinely do this, as flushing frequently increases file system overhead. For the same reason, you should not routinely set the log parameter at 31.

---

**See Also**

The following topics in the *Virtual AutoPilot Guide*:

- ❑ *Launching and Managing Multiple Script Playback*
- ❑ *Playback Timing*

## Virtual Script Player Command Line

You can launch the Virtual Script Player from LoadRunner, from Virtual Runner, or from the DOS command line. The command line must have entries that specify the user, the user's password, the environment, and the script name with a default extension of `.vsx` for any Virtual AutoPilot script, although this extension is not required.

The following four entries are required on the command line:

Command Line Abbreviation	Meaning	Sample Entry
-u	User	ce5791892
-p	PeopleSoft software user ID	-p pwd
-e	Environment	-e PDEV_VAP
-s	Script Name	-s voucherentry100.vsx

## Environment Initialization

The Virtual Script Player does not immediately begin playing a Virtual AutoPilot script upon launch from the DOS command line, from Virtual Runner, or from LoadRunner. In fact, the Virtual Script Player reads the script and runs events that generate a PeopleSoft software environment structure. The data that drives the generation of the environment comes from entries in the command line. For example, one user might create a Virtual AutoPilot script, but another user might play the script. During initialization, the Virtual Script Player passes in the user ID of the user playing the script, thereby creating the proper environment. Therefore, you can run the Virtual Script Player in an environment different from the one in which you or someone else created the Virtual AutoPilot script.

Environment initialization takes about 15 to 30 seconds. LoadRunner regards this passage as initializing time, while the DOS command line reads it as busy activity.

## Modes of Operation

The Virtual Script Player automatically detects whether you have launched a Virtual AutoPilot script from the DOS command line, from Virtual Runner, or from LoadRunner. If LoadRunner launches the script, the Virtual Script Player responds to stop/pause commands and sends transaction times and log output to LoadRunner. In addition, the Virtual Script Player completes a LoadRunner rendezvous just after it has initialized the system environment.

## Preprocessing of Valid Values List Data

The preprocessing capability of the Virtual User Tool works with the Virtual Script Editor and Virtual Script Player to use valid values lists during script playback. You must mark valid value lists because virtual playback requires the values contained in these lists as parameters for API database calls.

When a Virtual AutoPilot script specifies that a particular value originates in an AutoPilot valid values list, the Virtual Script Player reads the valid values list file. All valid values lists are identified by the extension `.atd`. Before the Virtual Script Player plays the script, it performs preprocessing that includes looking up the database values in the valid values list and storing them until they are required as parameters for API calls. When the Virtual Script Player runs the Virtual AutoPilot script, the stored list supplies the parameters needed for JDB or CallObject calls.

Preprocessing plays an important role in the Virtual AutoPilot scheme because it takes care of the lookup and load of the valid values that the Virtual Script Player needs for Virtual AutoPilot script execution. This ensures that the required values exist before playback. If the Virtual Script Player had to run database lookups at the time of script playback, the result would be artificial load on the database, which would, in turn, distort the simulation of activity that Virtual AutoPilot seeks to achieve.

## Valid Values List Processing

The Virtual Script Player defines the location of any valid value lists that are part of the Virtual AutoPilot script in the `vap.ini` file. The Virtual Script Player reads valid value lists that are 64K or smaller into memory. If the file is larger than 64K, the Virtual Script Player must read it from the file. During virtual playback, if the Virtual Script Player reaches the end of a valid values list, it starts

back at the beginning of the list, reuses the first value, and continues in sequence until virtual playback is complete.

## Date Formatting

The Virtual Script Player expects a certain format for date strings for valid value lists and for literal typed-in values from AutoPilot. Therefore, the Virtual Script Player supports different date formats that might appear in the Virtual AutoPilot script, including mm/dd/yyyy and Julian date strings (that is, 102343 or 12/09/2002).

---

### Caution

The Virtual Script Editor correctly formats date entries for literal values but not for date entries in valid value lists.

---

## Script Failure

Script failure might occur during the initialization process. For example, a branch event in the script might not refer to a valid event, or the events might not occur in the same thread. In the first example, the script fails before it is launched because the Virtual Script Player cannot validate the events. On initialization, the Virtual Script Player also validates function parameters. For example, a parameter such as Fetch might accept only 0 (zero) or 1 as values. If a different value is used, validation fails and, thus, the script fails before launching.

If the script fails during playback, the failure shuts down script processing. For most API calls, failure to return a success code causes the playback process to halt. The shutdown occurs without user intervention. LoadRunner, for example, returns a failure report, and the Virtual Script Player sends an error message to the log file, for example: `LoadRunner/Test Name/Local1/Subdirectory Name`. One subdirectory exists for every LoadRunner test session, which means that 50 simulated user test sessions produce 50 subdirectories.

If you launch the Virtual Script Player from a command line or from Virtual Runner and script failure occurs, no error message appears on the screen. You must open the log file that stores the test session results and examine the messages, a task you complete by searching on the keyword Error.

## Virtual Script Player Limitations

The overriding consideration for Virtual AutoPilot script playback is that client workstations must not impede the playback process. You must determine how many processes the workstation can realistically support, based on an analysis of workstation memory and CPU capability. Running either Task Manager or Performance Monitor can assess these capabilities.

Other Virtual Script Player limitations are hard-coded. If the Virtual Script Player gets a script that exceeds these limitations, you receive error messages that require a service pack to address. First, the Virtual Script Player supports up to 30 active user handles and 60 active request handles per session. Second, the Virtual Script Player can process only a certain number of status messages per second under LoadRunner. If the playback exceeds that number, some of the messages are lost, but the Virtual Script Player does not shut down.

---

## VSMEditor

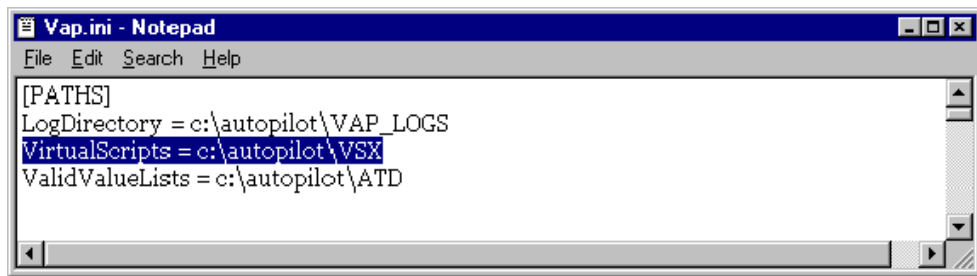
After you create a number of Virtual AutoPilot scripts, the VSMEditor allows you to concatenate any number of those scripts into a single master script. Concatenating single scripts into a single master script is advantageous because you can run a series of unrelated tasks during testing.

You control the VSMEditor from the VSMEditor form, which you access by clicking the VSMEditor executable.

### All Virtual Scripts List Box

The All Virtual Scripts list box contains all Virtual AutoPilot script files that you have created; these files have a `.vsx` extension. In addition, any master scripts that you have created appear in this list box; master scripts have a `.vsm` extension. The location of any Virtual AutoPilot script files that appear in the All Virtual Scripts list box is determined by the value of the `VirtualScripts` parameter of the `PATHS` section in the `vap.ini`.

You enter the path to the location of your virtual scripts to set the `VirtualScripts` parameter:



You can use any script in the All Virtual Scripts list box to create a master script. You create the script concatenation by choosing one of the scripts in the box and then holding down the Control key or the Shift key to choose additional scripts.

You click the Add button to add the files that you chose to the Master Script list box.

### Master Scripts List Box

The Master Scripts list box shows all the scripts that you have currently chosen for addition to a new `.vsm` (virtual script master) file. You can manipulate the script list in the Master Scripts list box by using the buttons adjacent to the box:

Button	Function
Remove	Deletes the chosen script from the Master Scripts list
Move Up and Move Down	Shift the position of the selected script in the list
Remove All	Deletes all scripts from the list
Save Master Scripts	Saves the list of scripts as a <code>.vsm</code> file

## VSM Files

The VSMEditor creates a `.vsm` text file when you save a master script. You can change these files only through the VSMEditor because the file contains a checksum value that verifies the file's integrity. The Virtual AutoPilot scripts always run in the sequence listed in the `.vsm` file. However, the first script to run is chosen randomly when the RandomStart parameter in the text file is set to 1.

The actual Virtual AutoPilot scripts are not included in the `.vsm` master file. Therefore, you should not delete scripts from the folder that contains the `.vsx` files.

---

## Virtual Runner

Virtual Runner controls the Virtual Script Player sessions on a single workstation and provides the following command and control functions for Virtual Script Player testing:

- Allows users to start one or more Virtual Script Player sessions on a single workstation
- Allows users to play multiple iterations of a single script
- Reports Player session status (pass/fail) to user
- Summarizes performance statistics over all Virtual Script Player sessions in a test

You use the action tools and the columns in the detail area of the Virtual Runner form to manage your Virtual Runner session.

## Player Session Columns

After you finish setting up the parameters for the Virtual Script Player session, Virtual Runner displays the names of the scripts that you want to run. Initially, the status of the script is Down, indicating that you have not yet run it.

After you run a test, Virtual Runner changes the status to indicate success or failure.

Each column displays information about your Virtual Script Player sessions. The following descriptions summarize the purpose of each player session column:

- The State column indicates the current state of the player session. For example, after you successfully execute a player session, this column displays the word Success.
- The Env column indicates the specified environment for the current session. The environment is specified using the Options button or when you use the Virtual Runner Wizard.
- The User column displays the User name that you specified using the Options button or the Virtual Runner Wizard.
- The Repeat column specifies the number of times the script is repeated when you execute the player session. You specify this parameter when you use the Virtual Runner Wizard.
- The Script column specifies the path and file name of the script for the current player session. You specify these parameters when you use the Virtual Runner Wizard.

## Actions Tools

You use the Actions tools to set up and launch a Virtual Runner session. You can choose the scripts that you want to run as well as the number of script playback iterations. In addition, following playback you can access a log that contains pertinent information about the playback session.

The Virtual Runner toolbar contains the following six buttons:

<b>Virtual Runner Toolbar Button</b>	<b>Button Function</b>
Option	Allows you to specify the user ID, password, and environment for the virtual playback session.
Wizard	Directs you through the process of specifying all the Virtual Script Player session parameters, including the number of scripts to run and the script playback iterations.
Run	Runs the virtual script playback session.
Log	Displays the Log Viewer screen, which provides information about the last completed Virtual Script Player session.
Report	Prints a copy of the Virtual Script Player session log information.
Close	Closes the Virtual Runner window after you have decided whether to save the results of the Virtual Script Player session.

# Creating Virtual Scripts

You use Virtual AutoPilot to create a script that simulates the activities of PeopleSoft software as it handles the workload generated by many users. To achieve this goal, you use two key components of PeopleSoft automated testing tools architecture:

- AutoPilot
- Virtual Script Editor

Using these two tools, you accomplish the following sequence of tasks to create a virtual script:

1. Create an AutoPilot script
2. Run the AutoPilot script with playback configured so that you can capture system and AutoPilot data
3. Import the event stream into the Virtual Script Editor
4. Create value links between source parameters of API calls and the target parameters of other API calls to ensure that usable data flows between API calls when you run the virtual script
5. Add loops to the Virtual AutoPilot script to account for repetitive processing, such as data retrieval
6. Generate the Virtual AutoPilot script, which the Virtual Script Player runs

After you create a virtual script, the Virtual Script Player runs the script. You use Virtual Runner or LoadRunner to manage the number of sessions, either from a single workstation or from multiple workstations.

---

## Capturing and Importing Test Results

AutoPilot allows you to create scripts that test PeopleSoft software applications. When you create a script, you can configure AutoPilot's playback function so that it captures and saves the results of your playback session, which it stores in the AutoPilot Playback Results Detail Table (F97214) as an event stream.

You can view the playback results in a variety of ways. You can view the event stream alone, you can view details of individual events, or you can view timing information about groups of events and thread identifiers, displayed in a horizontal bar graph.

The data that AutoPilot captures provides the raw material for your Virtual AutoPilot script. After you capture AutoPilot script data, you import it to the Virtual Script Editor so that you can prepare a virtual script.

# Capturing Test Results

To gather the raw data for a virtual script, you must first write and run an AutoPilot script and capture the results of the playback as an event stream. You use the Tools option in the menu bar of the AutoPilot form to set up the capture mechanism.

## ► To capture test results

---

*From your desktop or the appropriate directory, double-click the AutoPilot executable. Create the AutoPilot script or open an existing script.*

1. From the File menu, choose Open to open an AutoPilot script.
- 

### **Caution**

When you run the script, it must sign on to a PeopleSoft software environment. A script that does not include this signon does not function correctly in Virtual AutoPilot because it does not contain the data required for the Virtual Script Player to initialize the environment.

---

2. From the Tools menu, choose Options.
  3. On the Options form, choose the Playback tab.
  4. Choose the following options:
    - Save Results Data after Playback
    - Display Results Data after Playback
  5. In the Event Stream Capture Level portion of the Playback tab, choose the following:
    - Level 1 API calls
- 

### **Note**

If you want to capture more script playback events, choose the All API call levels option. Remember that you generate a much larger event stream if you choose this option.

---

6. Click OK.
7. Save the AutoPilot script.
8. In the AutoPilot menu bar, click Play and choose Play From Top.

AutoPilot runs the script. The Play From Top command generates test results for DENPCX (where DENPCX=the name of the machine on which AutoPilot resides). The AutoPilot Results form displays detailed information about the playback session.

9. Click Close to close the Test Results window.
10. Click File/Exit to close AutoPilot.

## Importing Test Results

After you have run an AutoPilot script and saved the playback results, you can import the event stream into the Virtual Script Editor. Importing the event stream allows you to use the Virtual Script Editor to forge value links between the source and target parameters of API calls; to identify, designate, and edit repetitive processing; and to generate a virtual script.

### ► To import playback results into the Virtual Script Editor

---

*From your desktop or the appropriate directory, double-click the Virtual Script Editor executable.*

1. On Virtual Script Editor, choose a script to import.
2. Click the Import button.

After the Virtual Script Editor imports the script, an APedit dialog box appears, confirming that the import was successful.

3. In the Virtual Script Editor dialog box, click OK.

---

#### **Caution**

If you attempt to import an AutoPilot script that you captured without a system signon, Virtual AutoPilot displays a warning issue.

If a message appears, you should recapture the data, making sure that you sign on to the system through AutoPilot. To do so, close PeopleSoft software, and then launch the AutoPilot script. AutoPilot handles your system signon.

---

## Viewing Test Results

After you successfully import the results of a script playback, the event stream appears in the detail area of the Virtual Script Editor form.

---

#### **Caution**

An exclamation point next to a start time (in the Start column) in a line of the event stream indicates that an error occurred during data capture.

If you find exclamation points in the event stream, you should investigate the possible causes for the error, and then edit and rerun the AutoPilot script.

---

To view the event stream alone, click the Details option in the toolbar. To view categories of playback events, thread activity, or both represented in a horizontal bar graph by duration and time of occurrence, click the Graph option, and then click the scroll bar button in the form to choose either View Graph by Message Type or View Graph by Thread ID. Click the Both option to view both the linear event stream and the horizontal bar graph representation.

## ► To view the event stream

---

*From your desktop or the appropriate directory, double-click the Virtual Script Editor executable.*

1. In the toolbar of the Virtual Script Editor form, click one of the following options:
  - Details
  - Graph
  - Both
2. Choose Details to view the linear event stream.
3. To view details about an individual event, click the event in the detail area, and then click import.

The Virtual Script Editor displays the details in the event stream detail pane.

4. To view in a horizontal bar graph categories of script playback events or the threads generated during playback, click the Graph option in the toolbar.
5. To choose the method of display for the graph, click the scroll bar button in the toolbar and choose one of the following:
  - View Graph by Msg Type
  - View Graph by Thread ID
6. To view both the event stream and the horizontal bar graph, choose the Both option in the toolbar.

---

## Editing the Virtual Script

After you import an event stream into the Virtual Script Editor, you can create a virtual script by completing the following two primary tasks:

- Adding loops
- Creating value links

After you finish these tasks, you generate the virtual script. The Virtual Script Editor passes the loop and value link information, as well as playback information that it stores automatically, to the Virtual Script Player, which runs the virtual script.

When you add loops, you define the number of data retrievals that Virtual AutoPilot performs when you run the virtual script. You can limit the number of loops, or you can ensure that the Virtual Script Player loops until no more data is available.

When you create value links, you ensure that data necessary to run the virtual script flows dynamically between parameters in API calls. For example, you must value link APIs that use next numbers so that the Virtual Script Player retrieves the appropriate next number during virtual playback. If you fail to value link the next number parameter in this scenario, the Virtual Script Player passes the same value used in the original script to the API parameter that requires it, which causes a duplicate key error. When you forge a value link, the Virtual Script Editor stores the parameter value in a variable, which ensures that the value changes each time you run the script, preventing duplicate keys and data contention.

The Virtual AutoPilot set also allows you to concatenate virtual scripts into a master script list using the VSMEditor. Using a master script enables you to test more than one script in a single virtual script playback session.

## Using the Find Feature

You use the Find feature in the Virtual Script Editor to search for parameters that you will need to value link to create the Virtual AutoPilot script. The Find feature consists of a control, in which you enter a search string. When the Virtual Script Editor finds a match, it displays a blue arrow in the event line in the pane that contains the match.

---

### Caution

Make sure you click inside the pane where you want to find information before you use the Find feature.

---

You can use the Virtual Script Editor Find feature to:

- Search for valid value list values to link. Enter a list value to the Find control.
- Find loops to process. Search for JDB Fetch calls.
- Find data dictionary aliases. Enter a data dictionary alias, such as AN8.

The Virtual Script Editor finds the first parameter with a data dictionary alias that matches your search criterion and marks it with an arrow.

---

### ► To use the find feature

---

1. Click inside the pane you want to search.
2. Type a value in the Find control.
3. Check the Case Sensitive button, if necessary.
4. Press Enter to run the search.

---

### Note

As you click a button to link or perform another task, you might lose the focus to the pane. Be sure to reset the focus to the pane you are searching, if necessary, by clicking inside the pane.

---

## Adding Loops

Loops in Virtual AutoPilot scripts simulate how PeopleSoft software functions when it performs inquiries. Without loops, the Virtual Script Player tries to fetch the same number of records that were retrieved during the original playback, regardless of selection criteria or available data. Loops also allow you to identify and reduce the number of events that must be value linked. Because of this advantage, you might want to generate loops before performing the value linking function.

## ► To add a loop

---

*From your desktop or the appropriate directory, double-click the Virtual Script Editor executable.*

1. Use the Find feature to search the event stream event pane for AUT: Press Button [Find].
2. From the Find statement in the event stream event pane, move the cursor down the list of events until you find a Fetch statement in the Message column.
3. Right-click and choose Add Loop.  
The Dynamic Loop Manager form appears.
4. On the Dynamic Loop Manager form, click one of the following options:
  - Loop until No Data  
Click if you want Virtual AutoPilot to exhaust data retrieval.
  - Loop X times  
Specifies the number of loops you want the script to perform.
5. Click Apply Loop to add the loop.
6. To undo the loop, find the loop, launch the Dynamic Loop Manager form, and click the Undo Loop button.

---

### Note

Virtual AutoPilot colors events inside a loop light gray in the event stream, source, and target panes. You do not need to consider these events when you perform value linking.

---

## Value Linking Parameters

Value linking allows data to flow from function to function within PeopleSoft software. For a Virtual AutoPilot script to accurately simulate system activities, it must not produce any duplicate key values in the system database. Therefore, for scripts that enter new data to the database or update existing data, at a minimum, you must value link all next number, job number, and batch number parameter values in the Virtual Script Editor. You can run simple inquiry scripts without any value linking, but these scripts might not accurately simulate system operations.

The Virtual Script Editor links some values automatically, but you must link others manually. To manually link values, you click the Link Parameters button on the toolbar. The Virtual Script Editor displays the Source Parameters pane and the Target Parameters pane. When you click an API call in the Target Parameters pane, the Virtual Script Editor displays in the Source Parameters pane only those API calls with a parameter that contains a value that matches the target parameter, if you do not choose the Show All option.

To run scripts accurately, you should always value link the parameters that:

- Pass the name of the machine on which you ran the original AutoPilot script.
- Reference the date on which the original AutoPilot script ran.
- Pass Next Numbers or serialized values (possibly labels of data items DOC, JOBS, MATH06, PYID, ICU).
- Use valid value list data. Linking these parameters ensures that the Virtual Script Player will use the .atd directory, where you store valid values list data as the source from which to retrieve data during virtual script playback.
- Contain the date that the script ran.

---

**Note**

You can use the Find feature to quickly find functions containing data to be linked. Click the column header to reorder the table (usually by label, value, or ID) to group like information.

---

► **To value link parameters**

---

*From your desktop or the appropriate directory, double-click the Virtual Script Editor executable.*

1. Import an AutoPilot script into the Virtual Script Editor.
2. Click the Link Parameters button in the toolbar.  
The Source Parameters pane and Target Parameters pane appear.
3. In the Target Parameters pane, choose a target parameter line item.  
The source parameters for that target display in the Source Parameters pane.

---

**Note**

Do not choose the Show All option in the Source Parameters pane because doing so causes the Virtual Script Editor to display all the API calls in the pane.

---

4. To link a single parameter line item, choose it and click the Link button.
5. To link all items in the script that match the source, target, label, and value parameters, choose a representative parameter line item in the source pane and click the Link All button.

---

**Note**

Some parameters in the Target Parameters pane do not have a value from a source parameter. You can mark these as Literal using the Mark Literal Button. If you do not want to see the parameters that you have marked as Literal, click Link in the menu bar and choose Filter Literals.

---

## Linking Values in Inquiry Scripts

Because an inquiry does not change or update any data in the system, you are not required to forge value links between parameters in inquiry scripts. However, you should value link parameters that contain valid values list data to ensure that the data changes during playback of the virtual script.

If your script contains valid values data, you can run the virtual script, change the data, and run it again to extend your stress testing. You can change the data in the list without creating new value links. During virtual script playback, the Virtual Script Editor passes the new valid values list data to the Virtual Script Player for use in the appropriate parameters.

### ► To link values in inquiry scripts

---

*From your desktop or the appropriate directory, double-click the Virtual Script Editor executable.*

1. In the Virtual AutoPilot Script Editor form, add any required loops to the Virtual AutoPilot Script.
2. Find valid values list data in the event stream.
3. Link all source parameters containing valid values list data to the appropriate target parameters.
4. Document the data dictionary aliases that the Virtual Script Editor links.

---

#### **Note**

You find data dictionary aliases in the Label column of the Source Parameters pane and the Target Parameters pane.

---

## Linking Values in Entry Scripts

Because entry scripts change or update system data, you are required to link values in entry scripts before you generate a virtual script. Value linking ensures that Virtual Script Player can pass values between parameters and that key parameter values change during virtual script playback, preventing record duplication.

### ► To link values in entry scripts

---

*From your desktop or the appropriate directory, double-click the Virtual Script Editor executable.*

1. In the Virtual Script Editor form, add any required loops to the Virtual AutoPilot Script.
2. Find and link any parameters that pass the machine name on which the AutoPilot script originally ran (these might be marked with CTID or MKEY data dictionary aliases or labels).
3. Find and link parameters that pass the date that the AutoPilot script originally ran.
4. Find and link parameters that pass Next Numbers or serialized values (possibly data dictionary aliases of DOC, JOBS, MATH06, PYID, and ICU).
5. Find and link parameters that pass valid values list values.

6. Document the data dictionary aliases that the Virtual Script Editor links.

## Generating the Virtual AutoPilot Script

When you press the Generate button, the Virtual Script Editor produces a virtual script, which the Virtual Script Player uses to simulate playback. A Script Log form appears following generation, summarizing the number of lines in the script and the number of errors, if any. You must generate an error-free script before you attempt to run it.

To generate the Virtual AutoPilot script for playback using the Virtual Runner program, the script that you want to generate must be open in the script list pane.

After Virtual AutoPilot generates the script, the Virtual Player Script Log form appears and displays information about the script generation, including the status (complete or incomplete), the number of lines, and the number of errors. If the Script Log form indicates that errors occurred during generation, you must investigate the error summaries that appear in the form and correct them by editing and rerunning the AutoPilot script, and then repeating the steps for creating a virtual script.

### ► To generate the Virtual AutoPilot script

---

*From your desktop or the appropriate directory, double-click the Virtual Script Editor executable.*

1. In the event stream event pane, click the Generate button.
2. Assign a file name and location to which you want to save the generated script and click OK to begin script generation.

After script generation completes, the Virtual Script Editor displays the Virtual Player Script Log form. If the summary reports any errors, you cannot use the script for virtual playback.

3. Click OK to close the Virtual Player Script Log form.

---

#### **Note**

After you generate a virtual script, it is static. Any script changes that you make in AutoPilot require re-editing and regeneration in Virtual User Tool. Thus, careful documentation of the editing process is critical to the production of repeatable results.

---

## Creating Master Scripts

Using the VSMEEditor tool, you can concatenate Virtual AutoPilot scripts into a single master script. Concatenation gives you another testing option: you can run test series of unrelated scripts.

### ► To create a master script

---

*From your desktop or the appropriate directory, double-click the VSMEEditor executable.*

1. In the All Virtual Scripts list box, choose the script files that you want to include in the master script.
2. When you have chosen all the virtual script text files that you want, click the Add button. VSMEEditor adds the script files to the Master Script list box.
3. Manipulate the list in the Master Script list box by using the buttons adjacent to the box to remove script files or to change their order.
4. When you have decided on the content and order of the master script, click the Save Master Script button.

The VSMEEditor saves the master script as a .vsm file. The file includes:

- Master script version
- Checksum value to verify file integrity
- RandomStart parameter (a value of 1 means that the first script to run is chosen randomly)
- List of Virtual AutoPilot Script files

# Running Virtual Scripts

After you generate one or more Virtual AutoPilot scripts, you are ready to execute playback to simulate multiple users running processes. If you want to simulate multiple users on a single workstation, you can launch the script either from a command line or from Virtual Runner.

Using Mercury Interactive's LoadRunner tool, you can also launch one or more Virtual AutoPilot script playback sessions on more than one workstation. LoadRunner manages the playback sessions. Using LoadRunner as your script playback manager allows you to more accurately simulate the actual stress that users in a business environment might impose on the system.

---

## Running Virtual Scripts from a Single Workstation

You can launch the Virtual Script Player from a command line on a single machine or from Virtual Runner, which manages virtual script playback, in order to simulate more than one user on a single workstation. The Virtual Script Player accesses the `.vsx` file that you create when you generate a virtual script on the Virtual Script Editor. After you run the script, you check the log files for errors.

## Running Virtual Scripts from a Command Line

The Virtual Script Player accesses the `.vsx` file generated by the Virtual Script Editor. You can launch the Virtual Script Player from a command line on a single machine or from a LoadRunner controller when you want to run virtual scripts on more than one workstation.

The command line must have entries specifying the user, the environment, and the script name. The following table summarizes the required entries on the command line:

Command Line Abbreviation	Description	Sample Entry
-u	User ID	-u JDE
-p	User password	-p JDE
-e	Environment	-e PRD733
-s	Script Name + number of script iterations to run	-s Script1.vsx

After you complete the virtual script run, you can review the log file for error messages. You set the path to the log file in the PATHS section of the Virtual AutoPilot initialization file.

When playback concludes, the `Virtual Script Player.exe` task disappears from the Task Manager window and a log in the `\\AutoPilot\VAP_Logs` directory displays any errors that were encountered. You can change the directory location in the `vap.ini` file.

### See Also

- *Virtual Script Player Initialization File Parameters* in the *Virtual AutoPilot Guide*

## ► To run virtual scripts from a command line

---

From the Start menu in Windows, choose Command Prompt from the Programs menu.

1. At the C: prompt, type the Virtual Script Player command with appropriate parameters. For example: `Virtual Script Player -u JDE -p JDE -e PRD733 -s5 script1.vsx`.
2. Press Enter to run the command.
3. To review the progress of the program, press Ctrl-Alt-Del to access the Windows Task Manager.

---

### Note

The Processes tab displays the executable (`Virtual Script Player.exe`) and the CPU activity associated with it. Otherwise, there is no indication of activity on the screen.

---

4. To search the Virtual AutoPilot log for errors, click the Search menu, choose Find, and search on the keyword *error*.

---

### Note

If errors occur, see *Debugging Virtual AutoPilot Scripts* in the *Virtual AutoPilot Guide*.

---

5. If the script contains valid values list data, change the data and play the script again.

## Running Virtual Scripts Using Virtual Runner

The Virtual Runner program allows you to manage the playback of virtual scripts. You use it to specify the script, the number of player sessions, and the number of iterations that you want to run in each session. You also specify the system environment against which you want to run the sessions.

After Virtual Runner finishes running the sessions, it displays the status of each test. You can view log information about a test by clicking the Log button.

You can expand the nodes in the Common Log Viewer form to see any error or warning messages that might have been issued during the Virtual Runner session. In addition to error and warning messages, the form displays:

- Name of the test
- Number of errors
- Number of warnings
- Status of the test
- Duration of the test
- Time of completion

You use the Virtual Runner log in conjunction with the system logs and more detailed VAP logs to help debug failed sessions.

## ► To run virtual scripts using Virtual Runner

---

*From your desktop or the appropriate directory, double-click the Virtual Runner executable.*

1. Click the Option button located on the Virtual Runner toolbar.  
The Option window appears.
2. Complete the following fields and then click OK:
  - User ID
  - Password
  - Environment  
The environment against which you want to run the test.
3. Click the Wizard button.  
The Virtual Runner Add Wizard - Step 1 of 3 form appears.
4. Click the Browse for script button to choose the script that you want to run.  
The Choose a Virtual Player Script to run form appears.
5. Choose the script that you want and click the Open button.  
The name of the script appears in the Choose a script to run field.
6. Specify the number of player sessions to run on the workstation.  
Example: enter 4 to run four scripts simultaneously.
7. Specify the number of virtual script session iterations to run.  
Example: enter 5 to run the script five iterations sequentially.
8. Click the Next button.  
The Virtual Runner Add Wizard - Step 2 of 3 form appears. If you entered information into the Option window, then the Wizard pulls that information into this window.
9. If you did not enter information into the Option window, enter your User ID, Password, and Environment.
10. Click Next.  
The Virtual Runner Add Wizard - Step 3 of 3 form appears.
11. If you want to add another script, click the button to add more scripts and repeat steps 4 through 10.
12. If you do not want to add additional scripts, click the Finish button to return to the Virtual Runner form.  
Virtual Runner displays the script or scripts that you chose and activates the Run button.

13. Click the Run button to begin script processing.

The main Virtual Runner screen displays the message *Starting Up*, indicating that the processing of the scripts has begun. The main Virtual Runner screen displays the message *Running* when Virtual Runner is processing the script or scripts. If the scripts successfully run, the screen displays the message *Success*. You are now ready to review the log file. If processing is not successful, a red failure message appears.

14. Click Log and click Yes to view the current test log.

The Common Log Viewer form appears.

15. Review the log for error and warning messages.

16. If the script contains valid values list data, change the data and play the script again.

### Prerequisite

- Before you can use Virtual Runner, you must cut the `vap.ini` file from

```
\\B9\system\Bin32
```

and paste it into

```
\\WINNT.
```

---

## Launching and Managing Multiple Script Playback

LoadRunner allows you to set up multiple workstations, each representing multiple users, from which you can launch playback sessions to simulate actual user load on the system. You provide LoadRunner with selected rendezvous points and transactions, which LoadRunner then reports to its controller. LoadRunner gathers and stores the results of each run. The LoadRunner controller workstation must have network connection to all of the workstations that are involved in the test, and the controller must run Windows NT.

### Defining a Script

You define the script that you want to play back so that LoadRunner can locate the Virtual Script Player and pass the Player the necessary script command line.

### Defining the Host Machine

After you have defined the scripts, you define the host machine for the LoadRunner test and the platform on which the test ran.

### Defining Virtual Users

After you define the script and the host machine, you define the virtual users who created the scripts that you want to run. You can define users individually or you can define a group as the virtual user.

## Setting Rendezvous Points

You set the rendezvous point that defines for LoadRunner the time at which all virtual scripts pause before the tool releases them for virtual playback.

## Gathering LoadRunner Results

The LoadRunner results directory typically has the following structure:

- VAPI (the test directory)
- User Name (from those defined in the Users window)
- Session Number Output.txt (the rerouted VAP\_log from the client workstation)

## Running Virtual Playback from the LoadRunner Controller

After you have prepared virtual script playback, you are ready to run the test from the LoadRunner controller.

# Special Considerations for Simulated Playback

The Virtual AutoPilot solves several simulated playback problems. All of the problems in one way or another revolve around the tool's ability to simulate accurately the workings of the PeopleSoft run-time engine. The Virtual Script Editor and the Virtual Script Player work together; the Virtual Script Editor stores key playback information and passes it to the Virtual Script Player, which in turn uses the information to assume the role of the run-time engine. This section explains important simulated playback problems and the ways that Virtual AutoPilot resolves them.

---

## Playback Timing

To accurately simulate system activities, Virtual AutoPilot must keep script events synchronized during playback. This presents a challenge because Virtual AutoPilot attempts to simulate multiple users who are stressing the server and the network, while the data upon which Virtual AutoPilot scripts are based is captured from a single user's script playback. This means that event start times and duration might change significantly during a virtual script playback session.

To meet this challenge, Virtual AutoPilot must solve two separate problems:

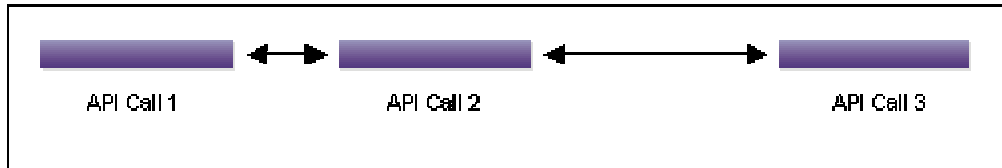
- Manage changes in the duration of individual API calls and the lengths of time between these calls within a single thread. This involves accurately simulating the process time required as the run-time engine handles user load.
- Handle timing differences that might affect interthread dependencies. These interdependencies occur when, for example, an API call in one thread has a data dependency on an API call in another thread.

## API Playback Timing

The goal of Virtual AutoPilot is to accurately simulate the stress that users place on the server and on the network. However, the AutoPilot script, which contains the data upon which the Virtual AutoPilot script is built, is not designed to create this stress. The time intervals between events in the AutoPilot script reflect the running of a single script against the run-time engine. When you run a Virtual AutoPilot script, the duration of events, and therefore the time intervals between those events, will likely change due to the server and network stress that the script is trying to simulate. The CPU power and memory capability of individual workstations can also affect the playback timing of Virtual AutoPilot scripts.

AutoPilot provides the base for the Virtual Script Player to time the execution of events during virtual script playback. When AutoPilot processes a script, it captures each kernel function call, and it captures the start time and duration of each API call. Therefore, the script contains the gaps of time between each call, which occur as the system carries out other processes. The API calls within a thread might be represented as blocks of time of various lengths with intervening spatial gaps that symbolize the time duration between each call, as illustrated below:

#### Auto Pilot Script Capture

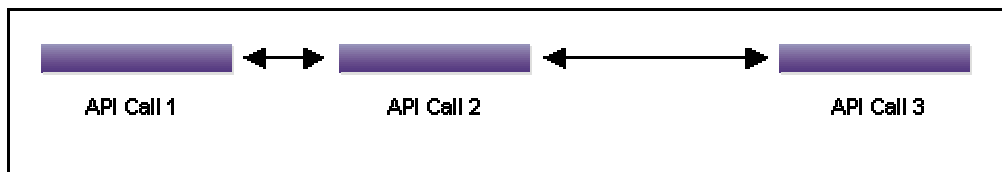


Preserving these chronological gaps during data capture provides the basis for simulating playback by many users, a situation that is likely to increase the length of time that is required to execute the same API calls.

For example, suppose that virtual playback on a single workstation simulates 10 users using a server that is not as powerful as the server that was in use when the AutoPilot script playback session originally occurred. In this scenario, the duration of API call is likely to lengthen, which could cause one API call to overlap another, halting playback.

However, since the event stream has preserved the *intervals* between each call, virtual playback can proceed, regardless of the duration of any or all of the calls within a thread, as illustrated below:

#### Stressed Playback in Virtual User Tool

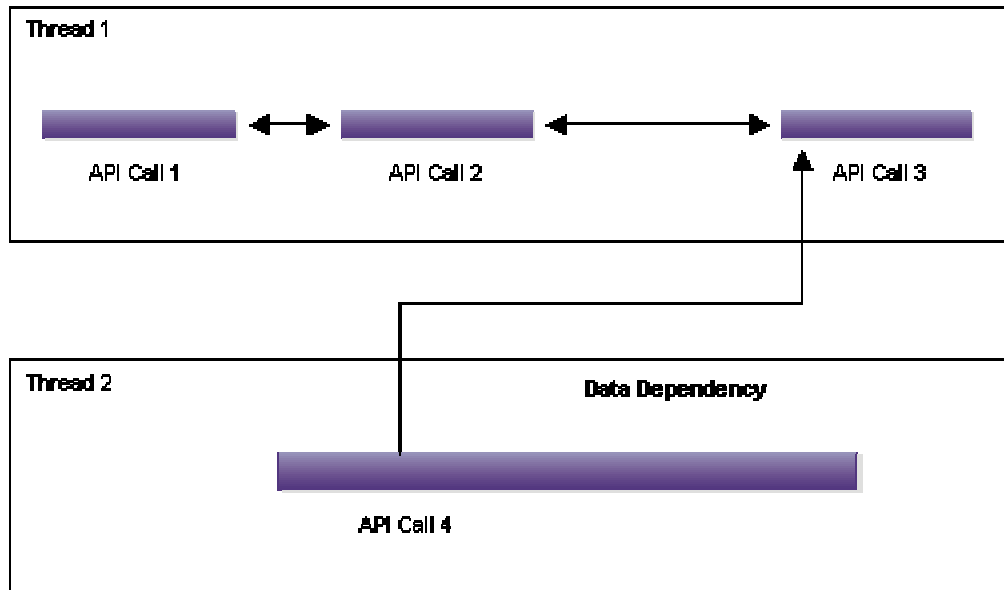


AutoPilot's ability to record the duration and length of time between API calls is also important because it can accurately determine the number of virtual users who can be simulated on a single workstation. For example, lengthy API calls might indicate an underpowered server, a workstation lacking the CPU and memory capability to handle the number of virtual user sessions that you desire, or an application bug. In each of these instances, you would likely scale back the number of users you want to simulate in a Virtual AutoPilot playback session.

## Interthread Timing

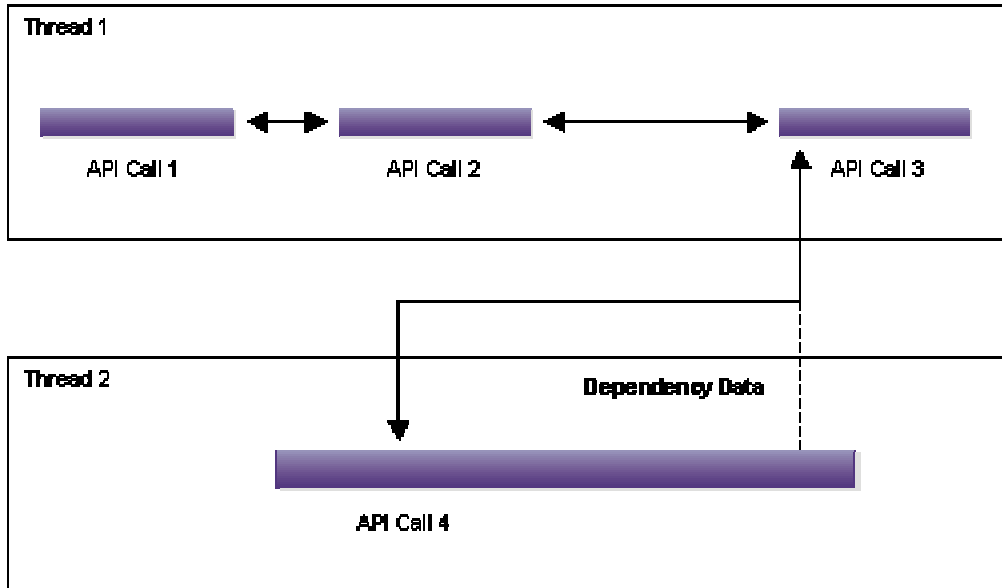
The Virtual Script Editor also plays a role in handling script playback timing so that Virtual AutoPilot can simulate a stressed environment. The run-time engine might, for example, create a thread that contains an API call with a data dependency on another call, which might, in turn, exist in a separate, asynchronously running thread. In this scenario, the run-time engine handles the processing tasks by noting that one API must finish before the data-dependent API can begin.

**Auto Pilot Script Capture**



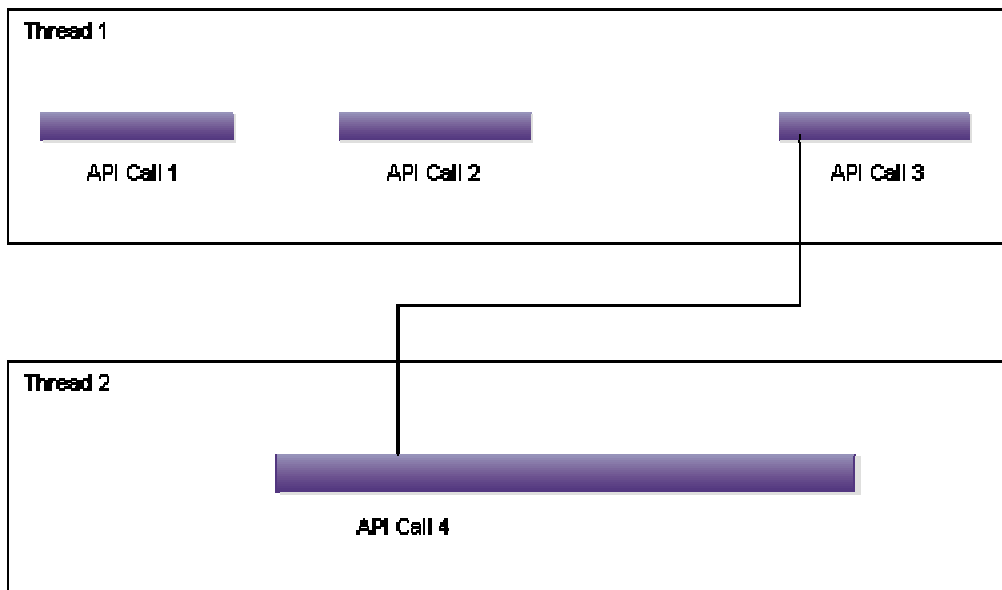
In a stressed environment, however, the duration of API calls might lengthen unpredictably. This might result in a data-dependent API call in one thread starting before the API upon which it depends has finished.

### Stressed Playback in Virtual User Tool



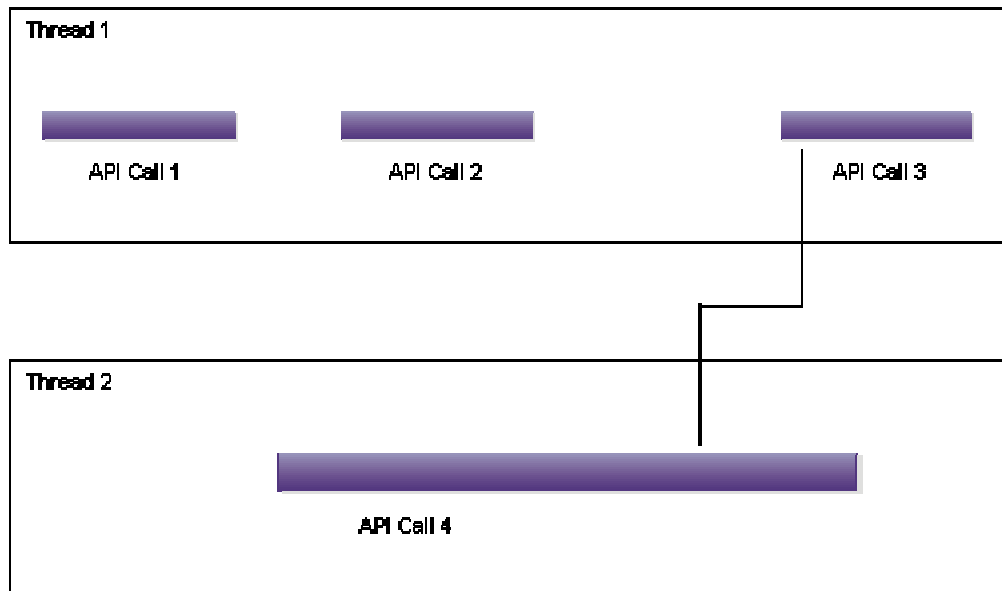
To deal with this potential problem, the Virtual Script Editor notes the data dependency when you forge value links between two API calls and preserves the timing interval between the calls.

### Auto Pilot Script Capture



When you run the virtual script, the Virtual Script Player increases the interval between APIs in one thread so that an API in one thread has time to complete before a data-dependent API in a thread running asynchronously to it is called.

## Stressed Playback in Virtual User Tool



In this way, Virtual AutoPilot preserves the necessary time interval that existed between the data-dependent calls when you originally ran the script.

Clearly, the Virtual Script Player, in this scenario, manipulates the time interval between API calls in the first thread. However, the manipulation represents an attempt to fairly simulate what the system does in reality. The run-time engine manages data-dependent APIs so that they can run without breaking the system. It is, therefore, appropriate that the Virtual Script Player, in assuming the role of the run-time engine, simulate the run-time engine's responsibility—for example, the delay of one API's completion based on its logical relationship to another API.

---

## Call Level

Some API calls invoke other API calls automatically within the same thread. Call level refers to an API call's position in the sequence of calls. For example, an EditLine business function might invoke a JDB Fetch call for a company number. In this example, the call level of the EditLine business function is 1, the call level of the JDB Fetch call is 2, and the AutoPilot event stream records the two separate API calls.

However, while the run-time engine handles two separate API calls in this example, the processing occurs seamlessly: the second call follows immediately from the first without additional input from the user. For this reason, a Virtual AutoPilot script contains only those API calls with a depth of 1. The Virtual Script Player automatically handles any API calls invoked by the original call, just as the run-time engine would.

This Virtual AutoPilot capability is important for playing scripts back in batch mode. If APIs with a call level greater than 1 were treated separately, repetitive processing would occur. Such repetition would not correctly simulate system processing.

The Virtual Script Editor provides a convenient way for you to view call level in the event stream. Each line that displays an initiating API call shows a call level of 1. Any calls that are invoked by the initiating call show a level of 2 or greater.

---

**Note**

The Virtual Script Editor displays API calls with a call level greater than 1 only if you choose the Capture Performance Statistics option when you configure playback in AutoPilot. If you do not wish to view call levels greater than 1, choose the Capture Virtual Script event stream option.

---

If you click the detail line of an API call that has a call level of 2 or greater, the event stream detail pane displays no parameters, meaning that you cannot value link any API call with a call level greater than 1. Therefore, no API calls with a call level greater than 1 appear in the Target Parameters pane.

---

## Synchronous and Asynchronous Calls

As part of simulating PeopleSoft software operations, Virtual AutoPilot must be able to manage synchronous and asynchronous API calls, an important management responsibility of the runtime engine. This ability ties into the Virtual Script Player's management of threads because an asynchronous call generates a separate thread.

A typical example of synchronous and asynchronous API call generation occurs when you enter data in a sales order line. You generate a synchronous call for each line edit; that is, the CallObject API for line 1 in a PeopleSoft grid precedes the CallObject API for line 2, and the CallObject API for line 2 does not occur until you have completed line 1. However, when you reach the end of a line, press the tab button, and proceed to line 2, you also generate an asynchronous API call that includes the data structure for the line that you just completed. The asynchronous CallObject API validates the data that you entered in line 1 through a series of related API calls. Meanwhile, you move ahead and begin entering sales order data in line 2.

The runtime engine manages this situation by generating a new thread for asynchronous calls and sending these calls to a queue to manage on a first-in, first-out (FIFO) basis. For example, you might enter 20 lines to the sales order entry grid. As you reach the end of each line and tab, the system will likely generate a new asynchronous call. Therefore, a number of asynchronous calls might queue for managing. When the runtime engine finishes processing the asynchronous calls, it stops the thread.

Virtual AutoPilot manages the simulation of asynchronous call management through the operation of each part of its architecture. The AutoPilot and PeopleSoft software hooks capture the timings of the synchronous and asynchronous calls that script playback generates. The Virtual Script Editor preserves the thread identifiers produced during playback, and the Virtual Script Player generates thread synchronization events in the Virtual AutoPilot script based on the temporal relationships among events in the captured event stream.

The Virtual Script Player also manages the threads generated during virtual playback. When virtual playback yields an asynchronous call, the Virtual Script Player queues the calls in a new thread and manages them on the same FIFO terms that the runtime engine uses, thereby managing interthread synchronization as well as event timing within threads.

Synchronous and asynchronous call management provides another example of the Virtual User Tool's ability to accurately simulate the system, thereby providing you with a realistic picture of network and server stress.

## See Also

- *Playback Timing* in the *Virtual AutoPilot Guide*

---

## Think Times

You insert wait periods while writing an AutoPilot script in an attempt to accurately simulate the way people use PeopleSoft software. You click the Wait Before Proceeding button and insert pauses into the playback. These pauses are in millisecond increments.

One possible reason for inserting wait periods in the script is to simulate the pauses that might occur as a user enters vouchers. A user might pause to answer the phone or tend to other tasks, and then return to making the entries.

The event stream generated during AutoPilot script playback records these wait times. They do not appear, however, with a label in the event stream pane if you import the event stream into the Virtual Script Editor. Rather, you recognize them by noting in the event stream pane the duration of time between the end of one event and the beginning of the next.

The Virtual AutoPilot script that you create contains the waits inserted in the original script, and the Virtual Script Player manages the delays during script playback. The inclusion of think times provides another element that helps Virtual AutoPilot simulate the PeopleSoft environment, which includes many users performing different tasks under a variety of circumstances.

You might want to analyze the event stream to see the length of time the events in the AutoPilot script took to complete. Think times that you insert into the script do not interfere with event duration analysis because the Virtual Script Editor's event graph does not reflect any wait times. If, for example, a five-second wait occurs between a CallObject API and an OpenTable API, the event graph displays only the amount of time required to run the APIs. Thus, you get a true picture of the time that the system required to process the API calls.

# Virtual AutoPilot Troubleshooting Tips and Techniques

You might encounter a script failure when you play back your Virtual AutoPilot script. Troubleshooting Virtual AutoPilot scripts consists of the following tasks.

1. Locate the source of the script failure, which might be in either Virtual AutoPilot or the system.
2. Run through a short list of script debugging techniques.

These techniques correct errors in business function and database API calls, transaction timing, and multiple playback sessions. You might also need to debug the system. In some cases, the problem lies in the original AutoPilot script or in application source code.

3. Review your AutoPilot script if you created it without first validating it through replay.

You cannot trace all failures of Virtual AutoPilot scripts to a single source, nor can you debug all scripts using a single method. In learning tips and techniques for troubleshooting Virtual AutoPilot scripts, you also learn the best solution to apply to a particular problem.

---

## Locating the Causes of Virtual AutoPilot Script Failures

The `vap.log` file contains messages about each Virtual AutoPilot script that you run. Therefore, it is the primary source of information about errors that might cause your script to fail. You set the message level in your Virtual AutoPilot initialization file. You should generally set the message parameter at 0, 1, 3, or 7 to minimize the number of messages that you collect. Setting the parameter higher causes slower playback performance and at least potentially skew playback results, thereby making performance analysis difficult. However, when you are attempting to find the source of a script failure, increasing the message level parameter temporarily can help you diagnose the problem.

If you fail to find the source of the script failure in Virtual User Tool, you can use several procedures to troubleshoot the system.

### See Also

- *Virtual Script Player Initialization File Parameters* in the *Virtual AutoPilot Guide*

## Finding Error Entries in the Virtual AutoPilot Log File

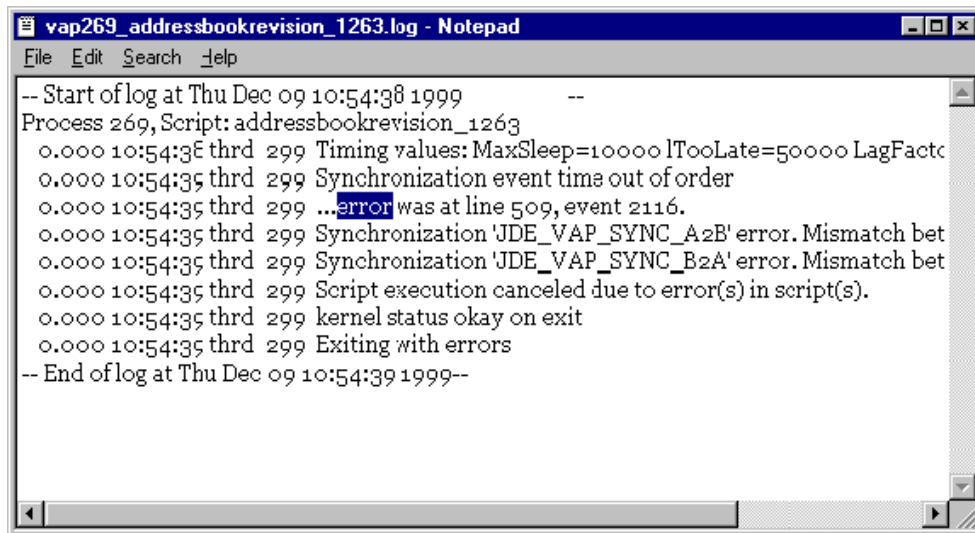
The Virtual Script Player sends an error message to the log file when a Virtual AutoPilot script fails during processing. If you launched the script from LoadRunner or from Virtual Runner, script failure halts the playback process, sending an instant signal that an error has occurred. However, if you launched the script from the DOS command line, you will not receive an error message. In either case, to isolate an error, go to the log file, which contains the test results, choose a test, open the text file, and search on the keyword *error*.

### ► To find entries in the Virtual AutoPilot log file

---

Locate and open the *vap.ini* file.

1. In the *vap.ini* file, go to the [PATHS] section to determine the location of the LogDirectory.
2. Follow the path to the LogDirectory.
3. Open the text file for the failed script using Notepad.
4. From the Edit menu, choose Find.
5. On the Find form, complete the following field with the word *error*.
  - Find what:



The screenshot shows a Notepad window titled "vap269\_addressbookrevision\_1263.log - Notepad". The window contains the following text:

```
-- Start of log at Thu Dec 09 10:54:38 1999 --
Process 269, Script: addressbookrevision_1263
0.000 10:54:35 thrd 299 Timing values: MaxSleep=100000|TooLate=50000 LagFacto
0.000 10:54:35 thrd 299 Synchronization event time out of order
0.000 10:54:35 thrd 299 ...error was at line 509, event 2116.
0.000 10:54:35 thrd 299 Synchronization 'JDE_VAP_SYNC_A2B' error. Mismatch bet
0.000 10:54:35 thrd 299 Synchronization 'JDE_VAP_SYNC_B2A' error. Mismatch bet
0.000 10:54:35 thrd 299 Script execution canceled due to error(s) in script(s).
0.000 10:54:35 thrd 299 kernel status okay on exit
0.000 10:54:35 thrd 299 Exiting with errors
-- End of log at Thu Dec 09 10:54:39 1999--
```

6. Note the line and event in which the error occurred.
7. Click Find Next to go to the next error.

### See Also

- *Virtual Script Player Initialization File Parameters* in the *Virtual AutoPilot Guide*

## Locating the Log File in the Event of Early Script Failure

The Virtual Script Player reads the location of the log directory out of the `vap.ini` text file. However, the script might fail before the Virtual Script Player has a chance to read the location. Therefore, when you go to the location of the log file that you specified as an initialization parameter, you will not find the test log. Despite the early failure, Virtual AutoPilot did log the errors. To find the test log, you go to the root of the drive that contains your log file and look for it there.

### ► To locate the Virtual AutoPilot log file in the event of early script failure

---

*Locate and open the `vap.ini` file.*

1. In the `vap.ini` file, determine the location of the LogDirectory.
2. Follow the path to the LogDirectory.
3. If the log that you are looking for is not in its usual location, go to the root of the drive and look for the log.

---

#### Note

If you do not find a log file in either location, you must examine your Virtual AutoPilot setup. Make sure that Virtual AutoPilot is installed completely and correctly.

---

## Setting the MessageLevel Parameter

You can set a message level parameter in the `vap.ini` file. This setting controls the kind and number of error messages that you receive in the `vap.log` file when you play back a Virtual AutoPilot script.

You might find very few messages in the log file as a result of setting the debug parameter too low. For example, if you set the parameter to 0 (zero), you will receive only a minimal number of messages. To increase the number of messages, go into the `vap.ini` file and change the MessageLevel parameter to 1, 3, 7, or 15. At each successive level, the Virtual Script Player writes more messages to the log.

### ► To set the MessageLevel parameter

---

*Locate and open the `vap.ini` file.*

1. In the `vap.ini` file, find the [Log] entry.
2. If the MessageLevel parameter is set lower than you want, change the setting.
3. Save your change and close the `vap.ini` file.

---

**Note**

If the Virtual Script Player crashes while you are running a script, you might find very few messages in the log file. This occurs because the Virtual Script Player did not flush the log file buffer, in which messages are stored, before the crash. You can prevent this by setting the message level parameter at 31. This parameter requires that the Virtual Script Player flush the log file buffer after each message. Remember, however, that system performance decreases when you set the message level at 31, so you should not leave it at that level permanently.

---

## Identifying an Environment Problem

If your Virtual AutoPilot script fails very early, even before the system completes its initial system logon, you might not be initializing the environment. In this case, you can troubleshoot system operations rather than Virtual AutoPilot operations. For example, you can try to log on to Explorer and run it through several sample tasks, such as opening an application. Use the same user ID, password, and environment name when you log onto the system that you used when logging on to the Virtual Script Player. You also can troubleshoot system errors, as these might also prevent you from replaying your Virtual AutoPilot script. If you have cleared any problems that might exist in running Explorer, try running your Virtual AutoPilot script again.

### Diagnosing an Environment Problem

Because Virtual User Tool's primary task is to simulate system operations, it must be able to initialize an environment at script playback time. For this to happen, the system itself must be initializing correctly. To exclude the Virtual Script Player as the source of script failure, you might attempt to sign on to the system to make sure that it is opening and running correctly.

#### ► To diagnose an environment problem

---

1. Close the Virtual Script Player.
2. Sign on to Explorer.
3. Perform several operations, such as accessing an application, changing forms, adding data, and so on.
4. If you are certain that the system is running correctly, rerun the script.

---

**Note**

Be sure to use the same user ID, password, and environment that you use when you log on to the Virtual Script Player.

---

## Investigating System Errors

Even if the system is initializing correctly, you might find errors that occur when you attempt to enter or edit data in an application. To isolate errors that occur in the system, you can turn on debugging and attempt to correct the errors.

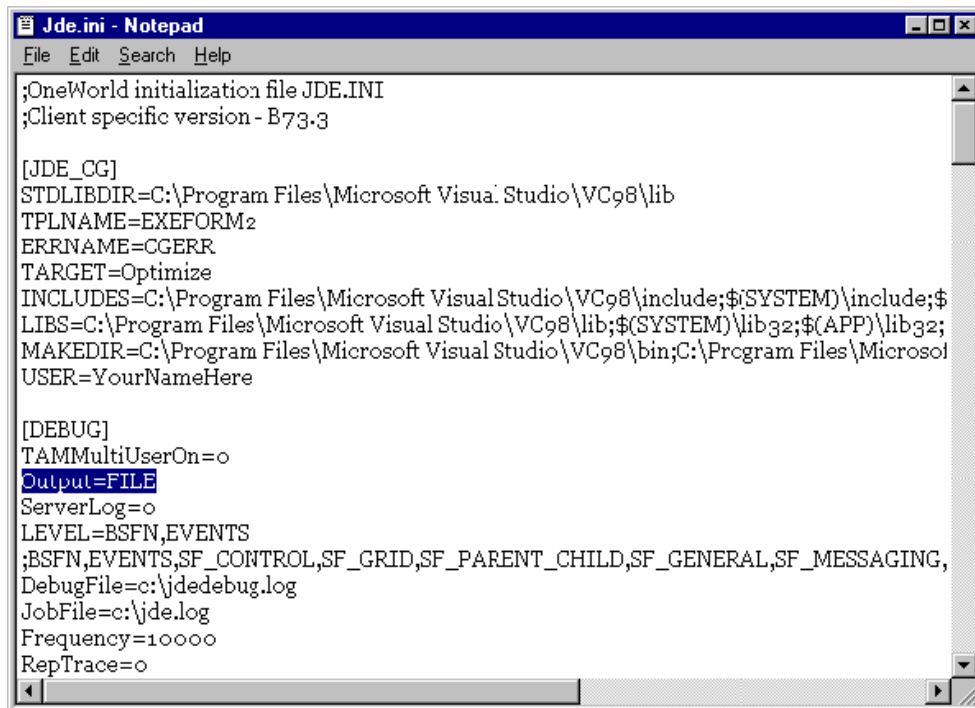
### ► To investigate system errors

---

1. Click the Windows Start menu and choose Run.
2. In the Open control of the Run form, type `JDE.INI`.

The `JDE.INI` file appears.

3. In the `JDE.INI` file, go to the `[DEBUG]` section.
4. Enter the Output parameter as `File`.



```
;OneWorld initialization file JDE.INI
;Client specific version - B73.3

[JDE_CG]
STDLIBDIR=C:\Program Files\Microsoft Visual Studio\VC98\lib
TPLNAME=EXEFORM2
ERRNAME=CGERR
TARGET=Optimize
INCLUDES=C:\Program Files\Microsoft Visual Studio\VC98\include;$(SYSTEM)\include;$(
LIBS=C:\Program Files\Microsoft Visual Studio\VC98\lib;$(SYSTEM)\lib32;$(APP)\lib32;
MAKEDIR=C:\Program Files\Microsoft Visual Studio\VC98\bin;C:\Program Files\Microsoft
USER=YourNameHere

[DEBUG]
TammMultiUserOn=0
Output=FILE
ServerLog=0
LEVEL=BSFN,EVENTS
;BSFN,EVENTS,SF_CONTROL,SF_GRID,SF_PARENT_CHILD,SF_GENERAL,SF_MESSAGING,
DebugFile=c:\jdedebug.log
JobFile=c:\jde.log
Frequency=10000
RepTrace=0
```

5. Run an AutoPilot script or access the system and run applications that are failing in the Virtual AutoPilot script.

---

#### Note

If error messages display in the status bar, click the stop sign. Read the error messages that appear. You can right-click error messages to display more troubleshooting information about each one.

---

6. Open the `jdedebug.log` file to evaluate any errors that occur.

---

**Caution**

Change the Output parameter in the [DEBUG] section of the JDE.INI file to NONE after you have corrected errors that prevent the Virtual AutoPilot script from functioning correctly.

---

---

## Debugging Virtual AutoPilot Scripts

If you have been troubleshooting problems with Virtual AutoPilot script playback but are still having trouble running the scripts, a business function call is likely causing the failure. You can review the log file to locate the source of the error, and you can identify the particular business function call that failed. You should have the message level in the vap.ini file set at 15 so that the log file displays parameter values.

You might also encounter problems that complicate your performance characterization efforts. For example, transaction information that you incorrectly or incompletely enter in an AutoPilot script might cause irregular transaction times in the Virtual AutoPilot script, thus making it difficult to draw accurate conclusions about system performance. In this case, you should troubleshoot the AutoPilot script, making sure that you have completely and accurately scripted input commands. If you modify the AutoPilot script, remember to run it again, capture the playback results, and reimport the event stream into the Virtual Script Editor.

Virtual AutoPilot also allows you to play back a script multiple times in succession, another important feature for performance characterization. However, doing so might cause playback to lock, again defeating your efforts to draw clearly and confidently characterize system performance. In this event, check your disk space to make sure you have enough to handle the testing.

If you have exhausted all of the debugging possibilities discussed here, you must turn your attention to debugging Explorer. Remember that if the same errors that appear in your Virtual AutoPilot script appear when you run the application in PeopleSoft, you likely have a system problem that you must debug. Your debugging efforts might include a call to PeopleSoft System Support.

You can gain additional insight into potential system problems by double-clicking the stop sign that appears in the status bar of a PeopleSoft form when an error occurs. When you perform this action, the system displays explanatory text, including possible causes and solutions, that helps you diagnose the source of the error. You can get additional troubleshooting information by setting the Output parameter in the JDE.INI file to FILE. Remember that doing so will degrade system performance, so you should return the Output parameter in the JDE.INI file to NONE after you have diagnosed and corrected any problems with the script.

## Displaying Business Function Parameters

Displaying the business function call parameters helps you to debug your Virtual AutoPilot scripts. To do so, you set the MessageLevel parameter in the vap.ini file at 15 or at 31. At this level, the log file displays all the input and output parameter values of the following:

- Business function API calls in the script
- Text of any error messages
- File name of the business function

- Line number in the source code that contains the error

► **To display business function parameters**

---

*Locate and open the vap.ini file.*

1. In the vap.ini file, find the [Log] entry.
2. Set the Message Level parameter at 15 or 31.
3. Save your change and close the vap.ini file.

---

**Caution**

Remember that you should not set the MessageLevel parameter permanently at 31 as this will cause performance to degrade. Leaving the MessageLevel parameter at 15 does not significantly degrade performance, but it can cause many messages and a great deal of text to accumulate in the log file. You should not leave the message level permanently set at 15, as doing so could consume a significant amount of disk space.

---

## Diagnosing Business Function Failures in PeopleSoft Explorer

Your scripts must run properly in Explorer before Virtual AutoPilot can run them properly. Therefore, you should determine early whether business function API calls are failing in the system when you run an AutoPilot script. To do so, you can turn on PeopleSoft debugging in the JDE.INI file.

---

**Note**

When you run an application, right-click and choose View System Log to view the jdedebug.log.

---

You might set breakpoints in the AutoPilot script after commands that initialize a business function API call, which will allow you to check the jdedebug.log at these key points.

By verifying the system's ability to process the commands in the AutoPilot script, you either pinpoint or exclude the system as a source of script failure. If it is causing the script failure, you work on debugging the system; conversely, if the business functions process properly when you run the script in the system, concentrate on finding the source of the script failure in the Virtual User Tool.

► **To diagnose business function failures in PeopleSoft**

---

*Locate and open the JDE.INI file.*

1. In the JDE.INI file, go to the [DEBUG] section and set the output parameter to FILE.
2. In the AutoPilot script pane, right-click a command line that follows a command that runs a business function (optional).
3. Click Toggle Breakpoint (optional).
4. Play back the AutoPilot script, either to the end or to a designated breakpoint.

5. Right-click inside a PeopleSoft form.
6. Choose View System Log.
7. Click File.
8. In the drop-down menu, choose `c:\jdedebug.log`.
9. Troubleshoot the `jdedebug.log` file, searching for business functions.

## Troubleshooting Value Linking Errors

For a Virtual AutoPilot script to run correctly, you must value link all required target parameters to the appropriate source parameters using the Virtual Script Editor. Failing to do so, or forging value links improperly, could cause your script to fail.

### Researching Value-Linking Errors in the Virtual Script Editor

A business function API call might fail when you run your Virtual AutoPilot script because you incompletely value linked the business function parameters in the event stream to the parameters in the Virtual AutoPilot script while you were working in the Virtual Script Editor.

Remember that you must value link any parameters that do not use constant values during script playback. If you do not value link these parameters, the script fails because, typically, the script playback creates duplicate keys.

Any of the following parameters could require value linking:

- Job number
- Document number
- Batch number
- Any parameters to which you assign values from a valid values list

The following parameters might frequently require value linking:

- Computer identification
- Those that require dates

After you examine the log file and perform necessary value linking that you might not have completed during script editing, you can rerun the script with the `MessageLevel` parameter in the `[LOG]` section of your `vap.ini` file set at 15. This setting allows you to capture parameter values and value substitutions in the log file.

### ► To research value linking errors in the Virtual Script Editor

---

*Locate and open the `vap.ini` file.*

1. Set the `MessageLevel` parameter at 15.
2. Run the Virtual AutoPilot script.
3. In the `vap.log` file, search for business function errors.
4. In the Virtual Script Editor, verify your value linking.

---

**Note**

Remember that you are required to provide value links for the following parameters:

- Job number
- Document number
- Batch number
- Any parameter that uses a value from a valid values list

The following parameters might frequently require value linking:

- Computer identification
  - Those that require dates
- 

5. Perform any necessary value linking in the Virtual Script Editor.
6. Rerun the Virtual AutoPilot script.
7. Recheck the `vap.log` file and look for business function API errors.

## Verifying That Value Linking Is Functioning

You can verify that Virtual AutoPilot is linking parameter values by creating valid values lists in your AutoPilot script. In the Virtual Script Editor, you value link any parameters that use values from the valid values list. The Virtual Script Player should link the values in the valid values lists to the appropriate parameters in the Virtual AutoPilot script during Virtual AutoPilot script playback.

To verify that Virtual AutoPilot performs the value linking, you can set your `MessageLevel` parameter at 15 and run the Virtual AutoPilot script. After you run the script, you search the log file for valid values list data, identify that data, and change the data in the `.atd` file, which stores your valid values list data.

When you replay the script, Virtual AutoPilot should use the new data from the valid values list. After you replay the Virtual AutoPilot script, you can search the log file again for valid values list data to make sure that the Virtual Script Player used the new data rather than any of the old values. If the Virtual Script Player used any of the old values, you must go back to the Virtual Script Editor and make sure you have sufficiently linked all of the values from the valid values lists to the appropriate parameters in the Virtual AutoPilot script.

### ► To verify that value linking is functioning

---

*Locate and open the `vap.ini` file.*

1. In the `vap.ini` file, set the `MessageLevel` parameter at 15.
2. Run the Virtual AutoPilot script.
3. Review the log file for valid values list data.

---

**Note**

You can search for valid values list data using the .atd extension. Verify that the values you expect are present and look for any error messages associated with the data.

---

4. In the `c:\.atd` file, change the valid values list data.
5. In the Virtual Script Editor, make sure that you have value-linked all of the new data in the valid values list to the correct parameters in the Virtual AutoPilot script.
6. Rerun the Virtual AutoPilot script.
7. Review the log file for old valid values list data.
8. If you find any of the old valid values list data, review the value linking in the Virtual Script Editor.

## Identifying and Correcting Variable Value-Linking Errors

Another type of value linking related error occurs if you declare a value in a AutoPilot script but do not set its value. In this case, if you value link the variable, the Virtual Script Editor registers errors in the script log during the virtual script generation process. To correct the errors, you must modify the AutoPilot script by setting the value of the variable.

► **To identify and correct variable value-linking errors**

---

*From your desktop or the appropriate directory, double-click the Virtual Script Editor executable.*

1. Choose a test and click the Generate button.
2. Review the Virtual Player Script Log form for validation error messages.
3. If validation error messages appear in the Virtual Script Log form, reopen the AutoPilot script.
4. Set a value for any declared variables that do not have a value in the Virtual Script Editor.
5. Save and rerun the AutoPilot script.

---

**Caution**

Be sure that playback remains configured to capture the virtual script event stream.

---

6. Re-import the event stream into the Virtual Script Editor and regenerate the Virtual AutoPilot script.

## Verifying the Validity of Virtual AutoPilot Script Data

Business function errors that occur in the Virtual AutoPilot script might be caused by data errors. Data errors occur because the environment against which you wrote the AutoPilot script differs from the environment against which you attempt to play back the Virtual AutoPilot script.

To verify that the data you use in the Virtual AutoPilot script is valid in the environment against you will run it, you can do either of the following:

- Update the values in your valid values list so that they will work in the environment.
- Replay the AutoPilot script in the environment against which you will run the Virtual AutoPilot script.

In this case, you will have to re-import the event stream into the Virtual Script Editor and regenerate a Virtual AutoPilot script by re-establishing value links.

### ► To verify the validity of data in the Virtual AutoPilot script

---

*Locate and open the `vap.ini` file.*

1. In the `vap.ini` file, set the `MessageLevel` parameter to 15.
2. Run the Virtual AutoPilot script.
3. Search the log file for business function errors.
4. Verify that the environment against which you wrote the AutoPilot script and against which you ran the Virtual AutoPilot script is the same.
5. If the two environments are different, recreate your valid values lists so that they contain values that are valid for the environment against which you are running the Virtual AutoPilot script.

---

#### Note

You can also replay the AutoPilot script in the same environment against which you are running the Virtual AutoPilot script. In that case, follow the next two steps.

---

6. Re-import the event stream into the Virtual Script Editor.
7. Regenerate a Virtual AutoPilot script by forging value links between the source and target parameters.

## Identifying and Correcting Duplicate Key Errors

JDB Insert and Update API calls might fail in the Virtual AutoPilot script because of duplicate key errors. These errors occur when you attempt to enter two records with the same value into a key column.

The duplicate key error prevents you from doing this. Failure to value link all the necessary parameters in the Virtual AutoPilot script could cause duplicate key errors. You can view updated and inserted JDB API parameter values in the Virtual Script Editor.

---

**Note**

Duplicate keys could also result from an application error.

---

---

**► To identify and correct duplicate key errors**

---

*Locate and open the JDE.INI file.*

1. In the `JDE.INI` file, go to the `[DEBUG]` section.
2. Change the `Output` parameter to `FILE`.
3. Play the AutoPilot script.
4. Locate and open the `jddebug.log` file.

If you have duplicate key errors, you will find them in the `jddebug.log` file.

5. Open the script in the Virtual Script Editor.
6. Check value linking for all JDB Insert and Update API calls.
7. When you are sure that you have value linked all JDB Insert and Update API calls, rerun the script.
8. If you continue to get duplicate key errors, review the application for errors that might be causing the problem.

## Rectifying Irregular Transaction Times

You measure transaction times by choosing events as start and endpoints in your AutoPilot script. For example, you might launch an application, move from one form to another by clicking the Add button, and then make entries to several header controls and grid columns in an active form before closing that form.

You might label that entire sequence of commands, from launching the application to closing the form, as a transaction. To see how efficiently the system manages this transaction, you label launching the application as the start of the transaction and closing the form as the end of the transaction. You also apply a name to the transaction and attach that name to the start and to the end. You use the Wait/Comment command in AutoPilot to insert the start and end of the transaction into the script and to apply a name to the transaction.

If you do not include both a start and an end time for the transaction, you might find irregular or inexplicable transaction times in the log, or you might find that the transaction fails. Failing to ensure that the name that you applied to the start of the transaction matches precisely the name that you applied to the end of the transaction, including capital letters and any special characters, might also cause irregular transaction times or transaction failures.

---

**Caution**

Virtual AutoPilot transaction timing accuracy has several limitations that make broad-based performance characterization assertions impossible. Accurate timings can be achieved only on a discrete workstation, while Virtual AutoPilot simulates server load.

---

## ► To rectify irregular transaction times

---

*From your desktop or the appropriate directory, double-click the AutoPilot executable.*

1. In the script pane of the AutoPilot form, place the insertion cursor directly above the command line that represents the start of the transaction.
2. In the menu bar, click Command and then choose Wait/Comment.
3. In the uncompleted Comment list of the AutoPilot command pane, enter Start, a space, and a name for the transaction.
4. Click the Insert button.

AutoPilot inserts a command line marking the start of the transaction.

5. Place the insertion cursor after the command line that represents the end of the transaction.
6. In the AutoPilot menu bar, click Command and then choose Wait/Comment.
7. In the Comment list of the AutoPilot command pane, enter End, a space, and a name for the transaction.

AutoPilot inserts a command line marking the end of the transaction.

---

### Caution

The name that you assign to the end of the transaction must exactly match the name that you assign to the start of the transaction.

---

8. Click the Insert button.

## Preventing Multiple Script Playback Problems

Virtual AutoPilot allows you to play back the same script in consecutive sessions or to simulate multiple users playing back scripts simultaneously. In either case, you must make sure that you have sufficient disk space to handle the load created by Virtual AutoPilot script playback, particularly if you plan to run a long test involving many playback iterations or simulation of a large number of users. If you do not have sufficient disk space, you might find that Virtual AutoPilot script playback locks up after only a few playbacks.

### Debugging Virtual Runner

If Virtual Runner fails immediately after you click the Run button, first check the `vapplayer.exe` path specified in the `vap.ini` file. The `vap.ini [COMMAND] binname` parameter specifies the full path of the `VAPPlayer.exe` file.

Virtual Script Player should operate the same whether you run under Virtual Runner control or from a command line. If it does not, you might be running two different copies of the `vapplayer.exe`. This might occur if the `vap.ini [COMMAND] binname` parameter is pointing to an old version of the Virtual Script Player. Make sure that `binname` parameter points to the correct drive and directory, and that you discard any old versions of the Virtual Script Player that you might have on your workstation.

If you set Virtual Script Player to run a virtual script multiple times in succession, and the script only runs a few times before locking up, you should review the available disk space. If you have set the `JDE.INI` error logging settings at a high level, the `jde.log` and `jddebug.log` can fill a disk very quickly. Make sure that enough free space is available on all relevant disk drives before you start a long test.

## Debugging LoadRunner

If you have set the `JDE.INI` or `vap.ini` error logging settings at a high level, and you run many virtual user sessions, the network might become saturated, communications between LoadRunner controller and the host machines might become scrambled, or both. You can address this problem by setting the `MessageLevel` parameter in the `vap.ini` files on all machines lower. This will decrease the volume of log file traffic.

The following table summarizes steps that you can take to minimize Virtual AutoPilot script playback problems:

Situation Affecting Playback	Possible Solution
<code>jde.log</code> and <code>jddebug.log</code> messages fill up disk quickly during Virtual AutoPilot script playback	In <code>[DEBUG]</code> section of the <code>JDE.INI</code> file, set <code>Output</code> parameter to <code>NONE</code>
Virtual AutoPilot log file fills with messages, consuming disk space	In <code>[LOG]</code> section of the <code>vap.ini</code> file, set <code>MessageLevel</code> parameter to 0, 1, 3, or 7

## Correcting Uninitialized User Handle Errors

An error labeled *Uninitialized User Handle* might cause your Virtual AutoPilot script to fail. This error occurs when you attempt to create a Virtual AutoPilot script using playback results that you obtained from the first run of a PeopleSoft application when just-in-time installation occurs, or when you have system debugging turned on when you capture the results of AutoPilot script playback.

### ► To correct uninitialized user handle errors

---

*From your desktop or the appropriate directory, double-click the Virtual User Tool executable.*

1. In Virtual User Tool, discard the results of the script generation attempt that failed.
2. In AutoPilot, rerun the script in the same environment that you created it.
3. Use the new results data to generate a new Virtual AutoPilot script.

# EnterpriseOne PeopleBooks Glossary

<b>“as of” processing</b>	A process that is run at a specific point in time to summarize item transactions.
<b>52 period accounting</b>	A method of accounting that uses each week as a separate accounting period.
<b>account site</b>	In the invoice process, the address to which invoices are mailed. Invoices can go to a different location or account site from the statement.
<b>active window</b>	The window that contains the document or display that will be affected by current cursor movements, commands, and data entry in environments that are capable of displaying multiple on-screen windows.
<b>ActiveX</b>	A technology and set of programming tools developed by Microsoft Corporation that enable software components written in different languages to interact with each another in a network environment or on a web page. The technology, based on object linking and embedding, enables Java applet-style functionality for Web browsers as well as other applications (Java is limited to Web browsers at this time). The ActiveX equivalent of a Java applet is an ActiveX control. These controls bring computational, communications, and data manipulation power to programs that can “contain” them—for example, certain Web browsers, Microsoft Office programs, and anything developed with Visual Basic or Visual C++.
<b>activity</b>	In Advanced Cost Accounting, an aggregation of actions performed within an organization that is used in activity-based costing.
<b>activity driver</b>	A measure of the frequency and intensity of the demands that are placed on activities by cost objects. An activity driver is used to assign costs to cost objects. It represents a line item on the bill of activities for a product or customer. An example is the number of part numbers, which is used to measure the consumption of material-related activities by each product, material type, or component. The number of customer orders measures the consumption of order-entry activities by each customer. Sometimes an activity driver is used as an indicator of the output of an activity, such as the number of purchase orders that are prepared by the purchasing activity. See also cost object.
<b>activity rule</b>	The criteria by which an object progresses from a given point to the next in a flow.
<b>actual cost</b>	Actual costing uses predetermined cost components, but the costs are accumulated at the time that they occur throughout the production process.
<b>adapter</b>	A component that connects two devices or systems, physically or electronically, and enables them to work together.
<b>add mode</b>	The condition of a form where a user can enter data into it.
<b>advanced interactive executive</b>	An open IBM operating system that is based on UNIX.
<b>agent</b>	A program that searches through archives or other repositories of information on a topic that is specified by the user.

<b>aging</b>	A classification of accounts by the time elapsed since the billing date or due date. Aging is divided into schedules or accounting periods, such as 0-30 days, 31-60 days, and so on.
<b>aging schedule</b>	A schedule that is used to determine whether a payment is delinquent and the number of days which the payment is delinquent.
<b>allegato IVA clienti</b>	In Italy, the term for the A/R Annual VAT report.
<b>allegato IVA fornitori</b>	In Italy, the term for the A/P Annual VAT report.
<b>application layer</b>	The seventh layer of the Open Systems Interconnection Reference Model, which defines standards for interaction at the user or application program level.
<b>application programming interface (API)</b>	A set of routines that is used by an application program to direct the performance of procedures by the computer's operating system.
<b>AS/400 Common</b>	A data source that resides on an AS/400 and holds data that is common to the co-existent library, allowing PeopleSoft EnterpriseOne to share information with PeopleSoft World.
<b>assembly inclusion rule</b>	A logic statement that specifies the conditions for using a part, adjusting the price or cost, performing a calculation, or using a routing operation for configured items.
<b>audit trail</b>	The detailed, verifiable history of a processed transaction. The history consists of the original documents, transaction entries, and posting of records and usually concludes with a report.
<b>automatic return</b>	A feature that allows a user to move to the next entry line in a detail area or to the first cell in the next row in several applications.
<b>availability</b>	The expression of the inventory amount that can be used for sales orders or manufacturing orders.
<b>available inventory</b>	The quantity of product that can be promised for sale or transfer at a particular time, considering current on-hand quantities, replenishments in process, and anticipated demand.
<b>back office</b>	The set of enterprise software applications that supports the internal business functions of a company.
<b>backhaul</b>	The return trip of a vehicle after delivering a load to a specified destination. The vehicle can be empty or the backhaul can produce less revenue than the original trip. For example, the state of Florida is considered a backhaul for many other states—that is, many trucking companies ship products into the state of Florida, but most of them cannot fill a load coming out of Florida or they charge less. Hence, trucks coming out of Florida are either empty or produce less revenue than the original trip.
<b>balance forward</b>	The cumulative total of inventory transactions that is used in the Running Balance program. The system does not store this total. You must run this program each time that you want to review the cumulative inventory transactions total.

<b>balance forward receipt application method</b>	A receipt application method in which the receipt is applied to the oldest or newest invoices in chronological order according to the net due date.
<b>bank tape (lock box) processing</b>	The receipt of payments directly from a customer's bank via customer tapes for automatic receipt application.
<b>base location</b>	[In package management] The topmost location that is displayed when a user launches the Machine Identification application.
<b>basket discount</b>	A reduction in price that applies to a group or "basket" of products within a sales order.
<b>basket repricing</b>	A rule that specifies how to calculate and display discounts for a group of products on a sales order. The system can calculate and display the discount as a separate sales order detail line, or it can discount the price of each item on a line-by-line basis within the sales order.
<b>batch job</b>	A job submitted to a system and processed as a single unit with no user interaction.
<b>batch override</b>	An instruction that causes a batch process to produce output other than what it normally would produce for the current execution only.
<b>batch process</b>	A type of process that runs to completion without user intervention after it has been started.
<b>batch program</b>	A program that executes without interacting with the user.
<b>batch version</b>	A version of a report or application that includes a set of user-defined specifications, which control how a batch process runs.
<b>batch/lot tracking</b>	The act of identifying where a component from a specific lot is used in the production of goods.
<b>batch/mix</b>	A manufacturing process that primarily schedules short production runs of products.
<b>batch-of-one processing</b>	A transaction method that allows a client application to perform work on a client workstation, and then submit the work all at once to a server application for further processing. As a batch process is running on the server, the client application can continue performing other tasks. See also direct connect, store-and-forward.
<b>binary large object (BLOB)</b>	A collection of binary data stored as a single entity in a [file].
<b>binder clip</b>	See paper clip.
<b>black products</b>	Products that are derived from the low or heavy end of the distillation process—for example, diesel oils and fuel oils. See also white products.
<b>blend note</b>	Document that authorizes a blending activity, and describes both the ingredients for the blend and the blending steps that occur.

<b>blend off</b>	Reworking off-specification material by introducing a small percentage back into another run of the same product.
<b>blind execution</b>	The mode of execution of a program that does not require the user to review or change the processing options set for the program, and does not require user intervention after the program has been launched.
<b>boleto</b>	In Brazil, the document requesting payment by a supplier or a bank on behalf of a supplier.
<b>bolla doganale</b>	VAT-Only Vouchers for Customs. In Italy, a document issued by the customs authority to charge VAT and duties on extra-EU purchasing.
<b>bookmark</b>	A shortcut to a location in a document or a specific place in an application or application suite.
<b>bordero &amp; cheque</b>	In Brazil, bank payment reports.
<b>broker</b>	A program that acts as an intermediary between clients and servers to coordinate and manage requests.
<b>BTL91</b>	In the Netherlands, the ABN/AMRO electronic banking file format that enables batches with foreign automatic payment instructions to be delivered.
<b>budgeted volume</b>	A statement of planned volumes (capacity utilization) upon which budgets for the period have been set.
<b>bunkering</b>	A rate per ton or a sum of money that is charged for placing fuel on board; can also mean the operation itself.
<b>business function</b>	An encapsulated set of business rules and logic that can normally be re-used by multiple applications. Business functions can execute a transaction or a subset of a transaction (check inventory, issue work orders, and so on). Business functions also contain the APIs that allow them to be called from a form, a database trigger, or a non-EnterpriseOne application. Business functions can be combined with other business functions, forms, event rules, and other components to make up an application. Business functions can be created through event rules or third-generation languages, such as C. Examples of business functions include Credit Check and Item Availability.
<b>business function event rule</b>	Encapsulated, reusable business logic that is created by using through event rules rather than C programming. Contrast with embedded event rule. See also event rule.
<b>business object library</b>	[In interoperability] The repository that stores EnterpriseOne business objects, which consist of Java or CORBA objects.
<b>business unit</b>	A financial entity that is used to track the costs, revenue, or both, of an organization. A business unit can also be defined as a branch/plant in which distribution and manufacturing activities occur. Additionally, in manufacturing setup, work centers and production lines must be defined as business units; but these business unit types do not have profit/loss capability.

<b>business view</b>	Used by EnterpriseOne applications to access data from database tables. A business view is a means for selecting specific columns from one or more tables with data that will be used in an application or report. It does not select specific rows and does not contain any physical data. It is strictly a view through which data can be handled.
<b>business view design aid (BDA)</b>	An EnterpriseOne GUI tool for creating, modifying, copying, and printing business views. The tool uses a graphical user interface.
<b>buy-back crude</b>	In foreign producing oil countries, that portion of the host government's share of "participation crude" which it permits the company holding a concession to "buy back."
<b>CAB</b>	In Italy, the bank branch code or branch ID. A five-digit number that identifies any agency of a specific bank company in Italy.
<b>cadastro de pessoas fisicas</b>	Cadastro de pessoas físicas. In Brazil, the federal tax ID for a person.
<b>category code</b>	A code that identifies a collection of objects sharing at least one common attribute.
<b>central object</b>	A software component that resides on a central server.
<b>central objects merge</b>	A process that blends a customer's modifications with the objects in a current release with objects in a new release.
<b>central server</b>	A computer that has been designated to contain the originally installed version of the software (central objects) for deployment to client computers.
<b>certificate input</b>	See direct input.
<b>certificate of analysis (COA)</b>	A document that is a record of all of the testing which has been performed against an item, lot, or both, plus the test results for that item and lot.
<b>change management</b>	[In software development] A process that aids in controlling and tracking the evolution of software components.
<b>change order</b>	In PeopleSoft, an addendum to the original purchase order that reflects changes in quantities, dates, or specifications in subcontract-based purchasing. A change order is typically accompanied by a formal notification.
<b>chargeback</b>	A receipt application method that generates an invoice for a disputed amount or for the difference of an unpaid receipt.
<b>chart</b>	EnterpriseOne term for tables of information that appear on forms in the software. See forms.
<b>check-in location</b>	The directory structure location for the package and its set of replicated objects. This location is usually \\deploymentserver\release\path_code\package\packagename. The subdirectories under this path are where the central C components (source, include, object, library, and DLL file) for business functions are stored.

<b>checksum value</b>	A computed value that depends on the contents of a block of data, and that is transmitted or stored with the data to detect whether errors have occurred in the transmission or storage.
<b>class</b>	[In object-oriented programming] A category of objects that share the same characteristics.
<b>clean cargo</b>	Term that refers to cargoes of gasoline and other refined products. See also dirty cargo.
<b>client access</b>	The ability to access data on a server from a client machine.
<b>client machine</b>	Any machine that is connected to a network and that exchanges data with a server.
<b>client workstation</b>	A network computer that runs user application software and is able to request data from a server.
<b>ClieOp03</b>	In the Netherlands, the euro-compliant uniform electronic banking file format that enables batches with domestic automatic direct debit instructions and batches with domestic payment instructions to be delivered.
<b>ClieOp2</b>	In the Netherlands, the uniform electronic banking file format that enables batches with domestic automatic direct debit instructions and batches with domestic payment instructions to be delivered.
<b>cluster</b>	Two or more computers that are grouped together in such a way that they behave like a single computer.
<b>co-existence</b>	A condition where two or more applications or application suites access one or more of the same database tables within the same enterprise.
<b>cold test</b>	The temperature at which oil becomes solid. Generally considered to be 5 degrees F lower than the pour point.
<b>commitment</b>	The number of items that are reserved to fill demand.
<b>common object request broker architecture</b>	An object request broker standard that is endorsed by the Object Management Group.
<b>compa-ratio</b>	An employee's salary divided by the midpoint amount for the employee's pay grade.
<b>component changeout</b>	See component swap.
<b>component object model (COM)</b>	A specification developed by Microsoft for building software components that can be assembled into programs or add functionality to existing programs running on Microsoft Windows platforms. COM components can be written in a variety of languages, although most are written in C++, and can be unplugged from a program at runtime without having to recompile the program.

<b>component swap</b>	In Equipment/Plant Management, the substitution of an operable component for one that requires maintenance. Typically, you swap components to minimize equipment downtime while servicing one of the components. A component swap can also mean the substitution of one parent or component item for another in its associated bill of material.
<b>conference room pilot environment</b>	An EnterpriseOne environment that is used as a staging environment for production data, which includes constants and masters tables such as company constants, fiscal date patterns, and item master. Use this environment along with the test environment to verify that your configuration works before you release changes to end-users.
<b>configurable network computing (CNC)</b>	An application architecture that allows interactive and batch applications that are composed of a single code base to run across a TCP/IP network of multiple server platforms and SQL databases. The applications consist of re-usable business functions and associated data that can be configured across the network dynamically. The overall objective for businesses is to provide a future-proof environment that enables them to change organizational structures, business processes, and technologies independently of each other.
<b>configurable processing engine</b>	Handles all “batch” processes, including reporting, Electronic Data Exchange (EDI) transactions, and data duplication and transformation (for data warehousing). This ability does not mean that it exists only on the server; it can be configured to run on desktop machines (Windows 95 and NT Workstation) as well.
<b>configuration management</b>	A rules-based method of ordering assemble-to-order or make-to-order products in which characteristics of the product are defined as part of the Sales Order Entry process. Characteristics are edited by using Boolean logic, and then translated into the components and routing steps that are required to produce the product. The resulting configuration is also priced and costed, based on the defined characteristics.
<b>configured item segment</b>	A characteristic of a configured item that is defined during sales order entry. For example, a customer might specify a type of computer hard drive by stating the number of megabytes of the hard drive, rather than a part number.
<b>consuming location</b>	The point in the manufacturing routing where a component or subassembly is used in the production process. In kanban processing, the location where the kanban container materials are used in the manufacturing process and the kanban is checked out for replenishment.
<b>contra/clearing account</b>	A G/L account used by the system to offset (balance) journal entries. For example, you can use a contra/clearing account to balance the entries created by allocations.
<b>contribution to profit</b>	Selling price of an item minus its variable costs.
<b>control table</b>	A table that controls the program flow or plays a major part in program control.
<b>control table workbench</b>	During the Installation Workbench process, Control Table Workbench runs the batch applications for the planned merges that update the data dictionary, user defined codes, menus, and user overrides tables.

<b>control tables merge</b>	A process that blends a customer's modifications to the control tables with the data that accompanies a new release.
<b>corrective work order</b>	A work order that is used to formally request unscheduled maintenance and communicate all of the details pertaining to the requested maintenance task.
<b>corrective work order</b>	A work order that is used to formally request unscheduled maintenance and communicate all of the details pertaining to the requested maintenance task.
<b>cost assignment</b>	Allocating resources to activities or cost objects.
<b>cost component</b>	An element of an item's cost—for example, material, labor, or overhead.
<b>cost object</b>	Any customer, product, service, contract, project, or other work unit for which you need a separate cost measurement.
<b>cost rollup</b>	A simulated scenario in which work center rates, material costs, and labor costs are used to determine the total cost of an item.
<b>costing elements</b>	The individual classes of added value or conversion costs. These elements are typically materials, such as raw and packaging; labor and machine costs; and overhead, such as fixed and variable. Each corporation defines the necessary detail of product costs by defining and tracking cost categories and subcategories.
<b>credit memo</b>	A negative amount that is used to correct a customer's statement when he or she is overcharged.
<b>credit notice</b>	The physical document that is used to communicate the circumstances and value of a credit order.
<b>credit order</b>	A credit order is used to reflect products or equipment that is received or returned so that it can be viewed as a sales order with negative amounts. Credit orders usually add the product back into inventory. This process is linked with delivery confirmation.
<b>cross segment edit</b>	A logic statement that establishes the relationship between configured item segments. Cross segment edits are used to prevent ordering of configurations that cannot be produced.
<b>crude oil assay</b>	A procedure for determining the distillation curve and quality characteristics of a crude oil.
<b>cumulative update</b>	A version of software that includes fixes and enhancements that have been made since the last release or update.
<b>currency relationships</b>	When converting amounts from one currency to another, the currency relationship defines the from currency and the to currency in PeopleSoft software. For example, to convert amounts from German marks to the euro, you first define a currency relationship between those two currencies.
<b>currency restatement</b>	The process of converting amounts from one currency into another currency, generally for reporting purposes. It can be used, for example, when many currencies must be restated into a single currency for consolidated reporting.

<b>current cost</b>	The cost that is associated with an item at the time a parts list and routing are attached to a work order or rate schedule. Current cost is based on the latest bill of material and routing for the item.
<b>customer pricing rules</b>	In Procurement, the inventory pricing rules that are assigned to a supplier. In Sales, inventory pricing rules that are assigned to a customer.
<b>D.A.S. 2 Reporting (DAS 2 or DADS 1)</b>	In France, the name of the official form on which a business must declare fees and other forms of remuneration that were paid during the fiscal year.
<b>data dictionary</b>	A dynamic repository that is used for storing and managing a specific set of data item definitions and specifications.
<b>data source workbench</b>	During the Installation Workbench process, Data Source Workbench copies all of the data sources that are defined in the installation plan from the Data Source Master and Table and Data Source Sizing tables in the Planner data source to the System - release number data source. It also updates the Data Source Plan detail record to reflect completion.
<b>data structure</b>	A description of the format of records in a database such as the number of fields, valid data types, and so on.
<b>data types</b>	Supplemental information that is attached to a company or business unit. Narrative type contains free-form text. Code type contains dates, amounts, and so on.
<b>datagram</b>	A self-contained packet of information that is forwarded by routers, based on their address and the routing table information.
<b>date pattern</b>	A period of time that is set for each period in standard and 52-period accounting and forecasting.
<b>DCE</b>	See distributed computing environment.
<b>DEB</b>	See déclaration d'échange de biens.
<b>debit memo</b>	In Accounts Payable, a voucher that is entered with a negative amount. Enter this type of voucher when a supplier sends you a credit so that you can apply the amount to open vouchers when you issue payment to the supplier.
<b>debit memo</b>	A form that is issued by a customer, requesting an adjustment of the amount, which is owed to the supplier.
<b>debit statement</b>	A list of debit balances.
<b>de-blend</b>	When blend off does not result in a product that is acceptable to customers. The further processing of product to adjust specific physical and chemical properties to within specification ranges. See also blend off.
<b>déclaration d'échange de biens (DEB)</b>	The French term that is used for the Intrastat report.
<b>delayed billing</b>	The invoicing process is delayed until the end of a designated period.

<b>delta load</b>	A batch process that is used to compare and update records between specified environments.
<b>denominated-in currency</b>	The company currency in which financial reports are based.
<b>deployment server</b>	A server that is used to install, maintain, and distribute software to one or more enterprise servers and client workstations.
<b>detail</b>	The specific information that makes up a record or transaction. Contrast with summary.
<b>detail information</b>	Information that primarily relates to individual lines in a sales or purchase order.
<b>direct connect</b>	A transaction method in which a client application communicates interactively and directly with a server application. See also batch-of-one immediate, store-and-forward.
<b>direct input</b>	The system calculates the net units when you enter gross volume, temperature, and gravity or density. This data is generally entered during product receiving from the certificate that is prepared by an independent inspector.
<b>direct ship orders</b>	A purchase order that is issued to a third-party supplier who designates the destination as the customer. A direct ship sales order is also created for the customer. Direct ship orders occur when a product is not available from a company-owned or company-operated source, so the system creates an order to ship the product from a third-party source directly to the customer. Sometimes referred to as a drop ship or third-party supply.
<b>direct usage</b>	Consumption of resources that are attributable to specific production runs because the resources were directly issued to the schedule/order.
<b>director</b>	An EnterpriseOne user interface that guides a user interactively through an EnterpriseOne process.
<b>dirty cargo</b>	Term that refers to crude oil cargoes or other non-refined petroleum cargoes. See also clean cargo.
<b>dispatch planning</b>	Efficient planning and scheduling of product deliveries. Considerations include: Dispatch groups Scheduled delivery date Scheduled delivery time Preferred delivery date Preferred delivery time Average delivery time for that geographical location Available resources Special equipment requirements at the product's source or destination.

<b>displacement days</b>	The number of days that are calculated from today's date by which you group vouchers for payment. For example, if today's date is March 10 and you specify three displacement days, the system includes vouchers with a due date through March 13 in the payment group. Contrast with pay-through date.
<b>display sequence</b>	A number that the system uses to re-order a group of records on the form.
<b>distributed computing environment (DCE)</b>	A set of integrated software services that allows software which is running on multiple computers to perform seamless and transparently to the end-users. DCE provides security, directory, time, remote procedure calls, and files across computers running on a network.
<b>distributed data processing</b>	Processing in which some of the functions are performed across two or more linked facilities or systems.
<b>distributed database management system (DDBMS)</b>	A system for distributing a database and its control system across many geographically dispersed machines.
<b>do not translate (DNT)</b>	A type of data source that must exist on the AS/400 because of BLOB restrictions.
<b>double-byte character set (DBCS)</b>	A method of representing some characters by using one byte and other characters by using two bytes. Double-byte character sets are necessary to represent some characters in the Japanese, Korean, and Chinese languages.
<b>downgrade profile</b>	A statement of the hierarchy of allowable downgrades. Includes substitutions of items, and meeting tighter specifications for those products with wider or overlapping specification ranges.
<b>DTA</b>	Datenträgeraustausch. A Swiss payment format that is required by Telekurs (Payserv).
<b>dual pricing</b>	To provide prices for goods and services in two currencies. During the euro transition period, dual pricing between the euro and Economic and Monetary Union (EMU) member currencies is encouraged.
<b>dynamic link library (DLL)</b>	A set of program modules that are designed to be invoked from executable files when the executable files are run, without having to be linked to the executable files. They typically contain commonly used functions.
<b>dynamic partitioning</b>	The ability to dynamically distribute logic or data to multiple tiers in a client/server architecture.
<b>economy of scale</b>	A phenomenon whereby larger volumes of production reduce unit cost by distributing fixed costs over a larger quantity. Variable costs are constant; but fixed costs per unit are reduced, thereby reducing total unit cost.
<b>edit mode</b>	A processing mode or condition where the user can alter the information in a form.
<b>edit rule</b>	A method that is used for formatting user entries, validating user entries, or both, against a predefined rule or set of rules.

<b>embedded event rule</b>	An event rule that is specific to a particular table or application. Examples include form-to-form calls, hiding a field that is based on a processing option value, or calling a business function. Contrast with business function event rule. See also event rule.
<b>employee work center</b>	A central location for sending and receiving all EnterpriseOne messages (system and user-generated), regardless of the originating application or user. Each user has a mailbox that contains workflow and other messages, including Active Messages. With respect to workflow, the Message Center is MAPI compliant and supports drag-and-drop work reassignment, escalation, forward and reply, and workflow monitoring. All messages from the message center can be viewed through EnterpriseOne messages or Microsoft Exchange.
<b>Emulator</b>	An item of software or firmware that allows one device to imitate the functioning of another.
<b>encapsulation</b>	The ability to confine access to and manipulation of data within an object to the procedures that contribute to the definition of that object.
<b>engineering change order (ECO)</b>	A work order document that is used to implement and track changes to items and resulting assemblies. The document can include changes in design, quantity of items required, and the assembly or production process.
<b>enhanced analysis database</b>	A database containing a subset of operational data. The data on the enhanced analysis database performs calculations and provides summary data to speed generation of reports and query response times. This solution is appropriate when external data must be added to source data, or when historical data is necessary for trend analysis or regulatory reporting. See also duplicated database, enterprise data warehouse.
<b>enterprise server</b>	A computer containing programs that collectively serve the needs of an enterprise rather than a single user, department, or specialized application.
<b>EnterpriseOne object</b>	A re-usable piece of code that is used to build applications. Object types include tables, forms, business functions, data dictionary items, batch processes, business views, event rules, versions, data structures, and media objects. See also object.
<b>EnterpriseOne process</b>	Allows EnterpriseOne clients and servers to handle processing requests and execute transactions. A client runs one process, and servers can have multiple instances of a process. EnterpriseOne processes can also be dedicated to specific tasks (for example, workflow messages and data replication) to ensure that critical processes do not have to wait if the server is particularly busy.
<b>EnterpriseOne web development computer</b>	A standard EnterpriseOne Windows developer computer with the additional components installed: Sun's JDK 1.1. JFC (0.5.1). Generator Package with Generator.Java and JDECOM.dll. R2 with interpretive and application controls/form.

<b>environment workbench</b>	During the Installation Workbench process, Environment Workbench copies the environment information and Object Configuration Manager tables for each environment from the Planner data source to the System release number data source. It also updates the Environment Plan detail record to reflect completion.
<b>equivalent fuel</b>	A barrel of equivalent fuel supplies six million BTUs of heat. Fuel gas quantities are usually calculated as equivalent fuel barrels in economic calculations for refinery operations.
<b>escalation monitor</b>	A batch process that monitors pending requests or activities, and restarts or forwards them to the next step or user after they have been inactive for a specified amount of time.
<b>ESR</b>	Einzahlungsschein mit Referenznummer. A pay slip with a reference number.
<b>event rule</b>	[In EnterpriseOne] A logic statement that instructs the system to perform one or more operations that are based on an activity that can occur in a specific application, such as entering a form or exiting a field.
<b>exit bar</b>	[In EnterpriseOne] The tall pane with icons in the left portion of many EnterpriseOne program windows.
<b>facility</b>	An entity within a business for which you want to track costs. For example, a facility might be a warehouse location, job, project, work center, or branch/plant. Sometimes referred to as a business unit.
<b>fast path</b>	[In EnterpriseOne] A command prompt that allows the user to move quickly among menus and applications by using specific commands.
<b>file handle</b>	A temporary reference (typically a number) that is assigned to a file which has been opened by the operating system and is used throughout the session to access the file.
<b>file server</b>	A computer that stores files to be accessed by other computers on the network.
<b>find/browse</b>	A type of form used to: Search, view, and select multiple records in a detail area. Delete records. Exit to another form. Serve as an entry point for most applications.
<b>firm planned order (FPO)</b>	A work order that has reached a user defined status. When this status is entered in the processing options for the various manufacturing programs, messages for those orders are not exploded to the components.
<b>fiscal date pattern</b>	A representation of the beginning date for the fiscal year and the ending date for each period in that year.
<b>fix/inspect</b>	A type of form used to view, add, or modify existing records. A fix/inspect form has no detail area.

<b>fixed quantity</b>	A term that indicates the bill of material relationship between a parent item and its components or ingredients. When a bill of material component has a fixed quantity relationship to its parent, the amount of the component does not change when the software calculates parts list requirements for different work order quantities. Contrast with variable quantity.
<b>flexible account numbers</b>	The format of account numbers for journal entries. The format that you set up must be the three segments:  Business unit.  Object.  Subsidiary.
<b>form design aid (FDA)</b>	The EnterpriseOne GUI development tool for building interactive applications and forms.
<b>form exit</b>	[In EnterpriseOne] An option that is available as a button on the Form Exit bar or as a selection in the Form menu. It allows users to open an interconnected form.
<b>form interconnection</b>	Allows one form to access and pass data to another form. Form interconnections can be attached to any event; however, they are normally used when a button is clicked.
<b>form type</b>	The following form types are available in EnterpriseOne:  Find/browse.  Fix/inspect.  Header detail.  Headerless detail.  Message.  Parent/child.  Search/select.
<b>form-to-form call</b>	A request by a form for data or functionality from one of the connected forms.
<b>framework</b>	[In object-oriented systems] A set of object classes that provide a collection of related functions for a user or piece of software.
<b>frozen cost</b>	The cost of an item, operation, or process after the frozen update program is run; used by the Manufacturing Accounting system.
<b>frozen update program</b>	A program that freezes the current simulated costs, thereby finalizing them for use by the Manufacturing Accounting system.
<b>globally unique identifier (GUI)</b>	A 16-byte code in the Component Object Model that identifies an interface to an object across all computers and networks.
<b>handle</b>	[In programming] A pointer that contains the address of another pointer, which, in turn, contains the address of the desired object.

<b>hard commitment</b>	The number of items that are reserved for a sales order, work order, or both, from a specific location, lot, or both.
<b>hard error</b>	An error that cannot be corrected by a given error detection and correction system.
<b>header</b>	Information at the beginning of a table or form. Header information is used to identify or provide control information for the group of records that follows.
<b>header information</b>	Information that pertains to the entire order.
<b>hover help</b>	A help function that provides contextual information or instructions when a cursor moves over a particular part of the interface element for a predefined amount of time.
<b>ICMS</b>	Imposto sobre circulação de mercadoria e serviços. In Brazil, a state tax that is applied to the movement of merchandise and some services.
<b>ICMS Substituto</b>	Imposto sobre circulação de mercadoria e serviços substituto. In Brazil, the ICMS tax that is charged on interstate transactions, or on special products and clients.
<b>ICMS Substituto-Markup</b>	See imposto sobre circulação de mercadoria e serviços substituto-markup.
<b>imposto de renda (IR)</b>	Brazilian income tax.
<b>imposto sobre produtos industrializados</b>	In Brazil, a federal tax that applies to manufactured goods (domestic and imported).
<b>imposto sobre services (ISS)</b>	In Brazil, tax on services.
<b>inbound document</b>	A document that is received from a trading partner using Electronic Data Interface (EDI). This document is also referred to as an inbound transaction.
<b>indented tracing</b>	Tracking all lot numbers of intermediates and ingredients that are consumed in the manufacture of a given lot of product, down through all levels of the bill of material, recipe, or formula.
<b>indexed allocations</b>	A procedure that allocates or distributes expenses, budgets, adjustments, and so on, among business units, based on a fixed percentage.
<b>indirect measurement</b>	Determining the quantity on-hand by: Measuring the storage vessels and calculating the content's balance quantity. or Theoretically calculating consumption of ingredients and deducting them from the on-hand balance.
<b>indirect usage</b>	Determining what should have been used by multiplying receipt quantity of the parent times the quantity per statement in the formula, recipe, or bill of material. This transaction typically affects both consumption on schedule as well as issue from on-hand balances.

<b>in-process rework</b>	<p>Recycling a semi processed product that does not meet acceptable standards. Further processing takes the product out of a given operation and sends it back to the beginning of that operation or a previous operation (for example, unreacted materials).</p> <p>Rework that is detected prior to receipt of finished goods and corrected during the same schedule run.</p>
<b>INPS withholding tax</b>	Instituto Nazionale di Previdenza Sociale withholding tax. In Italy, a 12% social security withholding tax that is imposed on payments to certain types of contractors. This tax is paid directly to the Italian social security office.
<b>inscrição estadual</b>	ICMS tax ID. In Brazil, the state tax ID.
<b>inscrição municipal</b>	ISS tax ID. In Brazil, the municipal tax ID.
<b>integrated toolset</b>	Unique to EnterpriseOne is an industrial-strength toolset that is embedded in the already comprehensive business applications. This toolset is the same toolset that is used by PeopleSoft to build EnterpriseOne interactive and batch applications. Much more than a development environment, however, the EnterpriseOne integrated toolset handles reporting and other batch processes, change management, and basic data warehousing facilities.
<b>integrity test</b>	A process that is used to supplement a company's internal balancing procedures by locating and reporting balancing problems and data inconsistencies.
<b>interbranch sales order</b>	A sales order that is used for transactions between branch/plants other than the selling branch/plant.
<b>Interoperability</b>	The ability of different computer systems, networks, operating systems, and applications to work together and share information.
<b>inventory pricing rule</b>	A discount method that is used for purchases from suppliers and sales to customers. The method is based on effectivity dates, up-to quantities, and a factor by which you can mark up or discount the price or cost.
<b>inventory turn</b>	The number of times that the inventory cycles, or turns over, during the year. A frequently used method to compute inventory turnover is to divide the annual costs of sales by the average inventory level.
<b>invoice</b>	An itemized list of goods that are shipped or services that are rendered, stating quantities, prices, fees, shipping charges, and so on. Companies often have their invoices mailed to a different address than where they ship products. In such cases, the bill-to address differs from the ship-to address.
<b>IP</b>	See imposto sobre produtos industrializados.
<b>IR</b>	See imposto de renda.
<b>IServer Service</b>	Developed by PeopleSoft, this Internet server service resides on the Web server and is used to speed up delivery of the Java class files from the database to the client.
<b>ISS</b>	See imposto sobre servicos.

<b>jargon</b>	An alternate data dictionary item description that EnterpriseOne or PeopleSoft World displays, based on the product code of the current object.
<b>java application server</b>	A component-based server that resides in the middle-tier of a server-centric architecture and provides middleware services for security and state maintenance, along with data access and persistence.
<b>JDBNET</b>	A database driver that allows heterogeneous servers to access each other's data.
<b>jde.ini</b>	A PeopleSoft file (or member for AS/400) that provides the runtime settings that are required for EnterpriseOne initialization. Specific versions of the file or member must reside on every machine that is running EnterpriseOne, including workstations and servers.
<b>JDE.LOG</b>	The main diagnostic log file of EnterpriseOne. Always located in the root directory on the primary drive. Contains status and error messages from the startup and operation of EnterpriseOne.
<b>JDEBASE Database Middleware</b>	PeopleSoft proprietary database middleware package that provides two primary benefits: <ol style="list-style-type: none"> <li>1. Platform-independent APIs for multidatabase access. These APIs are used in two ways: <ol style="list-style-type: none"> <li>a. By the interactive and batch engines to dynamically generate platform-specific SQL, depending on the data source request.</li> <li>b. As open APIs for advanced C business function writing. These APIs are then used by the engines to dynamically generate platform-specific SQL.</li> </ol> </li> <li>2. Client-to-server and server-to-server database access. To accomplish this access, EnterpriseOne is integrated with a variety of third-party database drivers, such as Client Access 400 and open database connectivity (ODBC).</li> </ol>
<b>JDECallObject</b>	An application programming interface that is used by business functions to invoke other business functions.
<b>JDEIPC</b>	Communications programming tools that are used by server code to regulate access to the same data in multiprocess environments, communicate and coordinate between processes, and create new processes.
<b>JDENET</b>	PeopleSoft proprietary middleware software. JDENET is a messaging software package.
<b>JDENET communications middleware</b>	PeopleSoft proprietary communications middleware package for EnterpriseOne. It is a peer-to-peer, message-based, socket-based, multiprocess communications middleware solution. It handles client-to-server and server-to-server communications for all EnterpriseOne supported platforms.
<b>just in time installation (JITI)</b>	EnterpriseOne's method of dynamically replicating objects from the central object location to a workstation.
<b>just in time replication (JITR)</b>	EnterpriseOne's method of replicating data to individual workstations. EnterpriseOne replicates new records (inserts) only at the time that the user needs the data. Changes, deletes, and updates must be replicated using Pull Replication.

<b>Kagami</b>	In Japan, summarized invoices that are created monthly (in most cases) to reduce the number of payment transactions.
<b>latitude</b>	The X coordinate of the location of an item in the warehouse. The system can use latitude, longitude, and height when suggesting locations for putaway, replenishment, and picking.
<b>laytime (or layhours)</b>	<p>The amount of time that is allotted to a tanker at berth to complete loading or discharging cargo. This time is usually expressed in running hours, and is fixed by prior agreement between the vessel owner and the company that is chartering the vessel. Laytime is stipulated in the charter, which states exactly the total of number of hours that are granted at both loading and unloading ports, and indicates whether such time is reversible. A statement of “Seventy-Two Hours, Reversible” means that a total of 72 hours is granted overall at both ports, and any time saved at one port can be applied as a credit at the other port.</p> <p>For example, if the vessel uses only 32 hours instead of 36 hours to load cargo, it can apply an additional four hours to the 36 hours allotted at the discharge port. Such considerations are important for purposes of computing demurrage.</p>
<b>leading zeros</b>	A series of zeros that certain facilities in PeopleSoft systems place in front of a value that is entered. This situation normally occurs when you enter a value that is smaller than the specified length of the field. For example, if you enter 4567 in a field that accommodates eight numbers, the facility places four zeros in front of the four numbers that you enter. The result appears as 00004567.
<b>ledger type</b>	A code that designates a ledger which is used by the system for a particular purpose. For example, all transactions are recorded in the AA (actual amounts) ledger type in their domestic currency. The same transactions can also be stored in the CA (foreign currency) ledger type.
<b>level break</b>	The position in a report or text where a group of similar types of information ends and another one begins.
<b>libro IVA</b>	Monthly VAT report. In Italy, the term for the report that contains the detail of invoices and vouchers that were registered during each month.
<b>line of business</b>	A description of the nature of a company’s work; also a tool to control the relationship with that customer, including product pricing.
<b>linked service type</b>	A service type that is associated with a primary service type. Linked service types can be cancelled, and the maintenance tasks are performed when the primary service type to which they are linked comes due. You can specify whether the system generates work orders for linked service types, as well as the status that the system assigns to work orders that have already been generated. Sometimes referred to as associated service types. See also primary service type and service type.
<b>livro razao</b>	In Brazil, a general ledger report.
<b>load balancing</b>	The act of distributing the number of processes proportionally to all servers in a group to maximize overall performance.

<b>location workbench</b>	During the Installation Workbench process, Location Workbench copies all locations that are defined in the installation plan from the Location Master table in the Planner data source to the System data source.
<b>log files</b>	Files that track operations for a process or application. Reviewing log files is helpful for troubleshooting problems. The file extension for log files is .LOG.
<b>logic data source</b>	Any code that provides data during runtime.
<b>logical compartment</b>	One of two ways that is identified in the transportation constants to display compartments on vehicles. Logical display numbers the compartments sequentially.  For example, if two vehicles are on a trip and each vehicle has three compartments, the logical display is 1,2,3,4,5,6.
<b>logical file</b>	A set of keys or indices that is used for direct access or ordered access to the records in a physical file. Several logical files can have different accesses to a physical.
<b>logical shelf</b>	A logical, not physical, location for inventory that is used to track inventory transactions in loan/borrow, or exchange agreements with other companies. See also logical warehouse.
<b>logical warehouse</b>	Not a physical warehouse containing actual inventory, but a means for storing and tracking information for inventory transactions in loan/borrow, or exchange agreements with other companies.
<b>longitude</b>	The Y coordinate of the location of an item in the warehouse. The system can use latitude, longitude, and height when suggesting locations for putaway, replenishment, and picking.
<b>LSV</b>	Lastschriftverfahren. A Swiss auto debit format that is required by Telekurs (Payserv).
<b>mail merge</b>	A mass-mail facility that takes names, addresses, and (sometimes) pertinent facts about recipients and merges the information into a form letter or a similarly basic document.
<b>mailmerge workbench</b>	[In EnterpriseOne] An application that merges Microsoft Word 6.0 (or higher) word-processing documents with EnterpriseOne records to automatically print business documents.
<b>main fuels</b>	Usually refers to bulk fuel products, but sometimes includes packaged products.
<b>maintenance loop</b>	See maintenance route.
<b>maintenance route</b>	A method of performing PMs for multiple pieces of equipment from a single preventive maintenance work order. A maintenance route includes pieces of equipment that share one or more identical maintenance tasks which can be performed at the same time for each piece of equipment. Sometimes referred to as maintenance loop.

<b>maintenance work order</b>	In PeopleSoft EnterpriseOne systems, a term that is used to distinguish work orders created for the performance of equipment and plant maintenance from other work orders, such as manufacturing work orders, utility work orders, and engineering change orders.
<b>manufacturing and distribution planning</b>	Planning that includes resource and capacity planning, and material planning operations. Resource and capacity planning allows you to prepare a feasible production schedule that reflects your demand forecasts and production capability. Material Planning Operations provides a short-range plan to cover material requirements that are needed to make a product.
<b>mapping</b>	A set of instructions that describes how one data structure passes data to another.
<b>master business function</b>	An interactive master file that serves as a central location for adding, changing, and updating information in a database.
<b>master business function</b>	A central system location for standard business rules about entering documents, such as vouchers, invoices, and journal entries. Master business functions ensure uniform processing according to guidelines that you establish.
<b>master table</b>	A database table that is used to store data and information that is permanent and necessary to the system's operation. Master tables might contain data such as paid tax amounts, supplier names, addresses, employee information, and job information.
<b>matching document</b>	A document that is associated with an original document to complete or change a transaction. For example, a receipt is the matching document of an invoice.
<b>media object</b>	An electronic or digital representation of an object.
<b>media storage objects</b>	Files that use one of the following naming conventions that are not organized into table format: Gxxx, xxxGT, or GTxxx.
<b>memory violation</b>	An error that occurs as the result of a memory leak.
<b>menu selection</b>	An option on a menu that initiates a software function directly.
<b>message center</b>	A central location for sending and receiving all EnterpriseOne messages (system- and user-generated), regardless of the originating application or user.
<b>messaging application programming interface (MAPI)</b>	An architecture that defines the components of a messaging system and how they behave. It also defines the interface between the messaging system and the components.
<b>metal content</b>	A series of properties of a blended product that help to determine its suitability for a prescribed purpose.
<b>metals management</b>	The process of maintaining information about the location and status of durable product containers such as liquid petroleum gas (LPG) cylinders.
<b>mobile inventory</b>	Inventory that is transferred from a depot to a barge or truck for milk-run deliveries.

<b>modal</b>	A restrictive or limiting interaction that is created by a given condition of operation. Modal often describes a secondary window that restricts a user's interaction with other windows. A secondary window can be modal with respect to its primary window or to the entire system. A modal dialog box must be closed by the user before the application continues.
<b>model work order</b>	For scheduled preventive maintenance or for a condition-based alert, a model work order functions as a template for the creation of other work orders. You can assign model work orders to service types and condition-based alerts. When the service type comes due or the alert is generated, the system automatically generates a work order that is based on information from the model work order.
<b>modeless</b>	Not restricting or limiting interaction. Modeless often describes a secondary window that does not restrict a user's interaction with other windows. A modeless dialog box stays on the screen and is available for use at any time, but also permits other user activities.
<b>multiple stocking locations</b>	Authorized storage locations for the same item number at locations, in addition to the primary stocking location.
<b>multitier architecture</b>	A client/server architecture that allows multiple levels of processing. A tier defines the number of computers that can be used to complete some defined task.
<b>named event rules (NER)</b>	Also called business function event rules. Encapsulated, re-usable business logic that is created by using event rules, rather than C programming.
<b>national language support (NLS)</b>	Mechanisms that are provided to facilitate internationalization of both system and application user interfaces.
<b>natureza da operação</b>	Transaction nature. In Brazil, a code that classifies the type of commercial transaction to conform to the fiscal legislation.
<b>negative pay item</b>	An entry in an account that indicates a prepayment. For example, you might prepay a supplier before goods are sent or prepay an employee's forecasted expenses for a business trip. The system stores these pending entries, assigning them a minus quantity as debit amounts in a designated expense account. After the prepaid goods are received or the employee submits an expense report, entering the actual voucher clears all of the negative pay items by processing them as regular pay items. Note that a negative pay item can also result from entering a debit memo (A/P) or a credit memo (A/R).
<b>net added cost</b>	The cost to manufacture an item at the current level in the bill of material. Thus, for manufactured parts, the net added cost includes labor, outside operations, and cost extras applicable to this level in the bill of material, but not materials (lower-level items). For purchased parts, the net added cost also includes the cost of materials.
<b>next status</b>	The next step in the payment process for payment control groups. The next status can be either WRT (write) or UPD (update).
<b>node</b>	A termination point for two or more communications links. A node can serve as the control location for forwarding data among the elements of a network or multiple networks, as well as performing other networking and, in some cases, local processing.

<b>non-inventory items</b>	See non-stock items.
<b>non-list price</b>	A price for bulk products that is determined by its own algorithms, such as a rolling average or commodity price plus.
<b>non-prime product</b>	A manufactured product with revenue potential that is less than the product planned for, or scheduled to be produced.
<b>non-stock items</b>	Items that the system does not account for as part of the inventory. For example, office supplies, or packaging materials can be non-stock items.
<b>nota fiscal</b>	In Brazil, a legal document that must accompany all commercial transactions.
<b>nota fiscal fatura</b>	In Brazil, a nota fiscal and invoice information.
<b>notula</b>	In Italy, the process whereby a business does not recognize value added tax until the payment of a voucher.
<b>object configuration manager (OCM)</b>	EnterpriseOne's object request broker and the control center for the runtime environment. It keeps track of the runtime locations for business functions, data, and batch applications. When one of these objects is called, the Object Configuration Manager directs access to it by using defaults and overrides for a given environment and user.
<b>object embedding</b>	When an object is embedded in another document, an association is maintained between the object and the application that created it; however, any changes made to the object are also only kept in the compound document. See also object linking.
<b>object librarian</b>	A repository of all versions, applications, and business functions that are reusable in building applications.
<b>object linking</b>	When an object is linked to another document, a reference is created with the file in which the object is stored, as well as with the application that created it. When the object is modified, either from the compound document or directly through the file in which it is saved, the change is reflected in that application as well as anywhere it has been linked. See also object embedding.
<b>object linking and embedding (OLE)</b>	A technology for transferring and sharing information among applications by allowing the integration of objects from diverse applications, such as graphics, charts, spreadsheets, text, or an audio clip from a sound program. OLE is a compound document standard that was developed by Microsoft Corporation. It enables you to create objects with one application, and then link or embed them in a second application. Embedded objects retain their original format and links to the application that created them. See also object embedding, object linking.
<b>object management workbench (OMW)</b>	The change management system that is used for EnterpriseOne development.

<b>object-based technology (OBT)</b>	A technology that supports some of the main principles of object-oriented technology: Classes. Polymorphism.I Inheritance. Encapsulation.
<b>object-oriented technology (OOT)</b>	Brings software development past procedural programming into a world of re-usable programming that simplifies development of applications. Object orientation is based on the following principles: Classes. Polymorphism.I Inheritance. Encapsulation.
<b>offsetting account</b>	An account that reduces the amount of another account to provide a net balance. For example, a credit of 200 to a cash account might have an offsetting entry of 200 to an A/P Trade (liability) account.
<b>open database connectivity (ODBC)</b>	Defines a standard interface for different technologies to process data between applications and different data sources. The ODBC interface comprises set of function calls, methods of connectivity, and representation of data types that define access to data sources.
<b>open systems interconnection (OSI)</b>	The OSI model was developed by the International Standards Organization (ISO) in the early 1980s. It defines protocols and standards for the interconnection of computers and network equipment.
<b>order detail line</b>	A part of an order that contains transaction information about a service or item being purchased or sold, such as quantity, cost, price, and so on.
<b>order hold</b>	A flag that stops the processing of an order because it has exceeded the credit or budget limit, or has another problem.
<b>order-based pricing</b>	Pricing strategy that grants reductions in price to a customer. It is based upon the contents and relative size (volume or value) of the order as a whole.
<b>outbound document</b>	A document that is sent to a trading partner using EDI. This term is also referred to as an outbound transaction.

<b>outturn</b>	<p>The quantity of oil that is actually received into a buyer's storage tanks when a vessel is unloaded. For various reasons (vaporization, clingage to vessel tank walls, and so on), the amount of a product pumped into shore tankage at unloading is often less than the quantity originally loaded onto the vessel, as certified by the Bill of Lading. Under a delivered or CIF outturn transaction, the buyer pays only for the barrels actually "turned out" by the vessel into storage.</p> <p>When a buyer is paying CIF Bill of Lading figures, a loss of 0.5% of total cargo volume is considered normal. Losses in excess of 0.5%, however, are either chargeable to the seller or are covered by specialized insurance that covers partial, as well as total, loss of the cargo.</p>
<b>overhead</b>	<p>In the distillation process, that portion of the charge that leaves the top of the distillation column as vapor. This definition is strictly as it relates to ECS.</p>
<b>override conversion method</b>	<p>A method of calculating exchange rates that is set up between two specific currencies. For those specific currencies, this method overrides the conversion method in General Accounting Constants and does not allow inverse rates to be used when calculating currency amounts.</p>
<b>package / package build</b>	<p>A collection of software that is grouped into a single entity for modular installation. EnterpriseOne objects are installed to workstations in packages from the deployment server. A package can be compared to a bill of material or kit that indicates the necessary objects for that workstation and where the installation program can find them on the deployment server. It is a point-in-time "snapshot" of the central objects on the deployment server.</p>
<b>package location</b>	<p>The directory structure location for the package and its set of replicated objects. This location is usually \\deployment server\release\path_code\package\ package name. The replicated objects for the package are placed in the subdirectories under this path. This location is also where the package is built or stored.</p>
<b>package workbench</b>	<p>During the Installation Workbench process, Package Workbench transfers the package information tables from the Planner data source to the System - release number data source. It also updates the Package Plan detail record to reflect completion.</p>
<b>packaged products</b>	<p>Products that, by their nature, must be delivered to the customer in containers which are suitable for discrete consumption or resale.</p>
<b>pane/panel</b>	<p>A resizable subarea of a window that contains options, components, or other related information.</p>
<b>paper clip</b>	<p>An icon that is used to indicate that a media object is attached to a form or record.</p>
<b>parent/child form</b>	<p>A type of form that presents parent/child relationships in an application on one form:</p> <p>The left portion of the form presents a tree view that displays a visual representation of a parent/child relationship.</p> <p>The right portion of the form displays a detail area in browse mode. The detail area displays the records for the child item in the tree.</p> <p>The parent/child form supports drag and drop functionality.</p>

<b>parent/child relationship</b>	See parent/component relationship.
<b>parent/component relationship</b>	<p>1. In Capital Asset Management, the hierarchical relationship of a parent piece of equipment to its components. For example, a manufacturing line could be a parent and the machinery on the line could be components of the line. In addition, each piece of machinery could be a parent of still more components.</p> <p>2. In Product Data Management, a hierarchical relationship of the components and subassemblies of a parent item to that parent item. For example, an automobile is a parent item; its components and subassemblies include: engine, frame, seats, and windows.</p> <p>Sometimes referred to as parent/child relationship.</p>
<b>partita IVA</b>	In Italy, a company fiscal identification number.
<b>pass-through</b>	A process where data is accepted from a source and forwarded directly to a target without the system or application performing any data conversion, validation, and so on.
<b>pay on consumption</b>	The method of postponing financial liability for component materials until you issue that material to its consuming work order or rate schedule.
<b>payment group</b>	A system-generated group of payments with similar information, such as a bank account. The system processes all of the payments in a payment group at the same time.
<b>PeopleSoft database</b>	See JDEBASE Database Middleware.
<b>performance tuning</b>	The adjustments that are made for a more efficient, reliable, and fast program.
<b>persistent object</b>	An object that continues to exist and retains its data beyond the duration of the process that creates it.
<b>pervasive device</b>	A type of intelligent and portable device that provides a user with the ability to receive and gather information anytime, from anywhere.
<b>planning family</b>	A means of grouping end items that have similarity of design or manufacture.
<b>plug-in</b>	A small program that plugs into a larger application to provide added functionality or enhance the main application.
<b>polymorphism</b>	A principle of object-oriented technology in which a single mnemonic name can be used to perform similar operations on software objects of different types.
<b>portal</b>	A Web site or service that is a starting point and frequent gateway to a broad array of on-line resources and services.
<b>Postfinance</b>	A subsidiary of the Swiss postal service. Postfinance provides some banking services.

<b>potency</b>	Identifies the percent of an item in a given solution. For example, you can use an 80% potent solution in a work order that calls for 100% potent solution, but you would use 25% more, in terms of quantity, to meet the requirement ( $100 / 80 = 1.25$ ).
<b>preference profile</b>	The ability to define default values for specified fields for a user defined hierarchy of items, item groups, customers, and customer groups. In Quality Management setup, this method links test and specification testing criteria to specific items, item groups, customers, or customer groups.
<b>preflush</b>	A work order inventory technique in which you deduct (relieve) materials from inventory when the parts list is attached to the work order or rate schedule.
<b>preventive maintenance cycle</b>	The sequence of events that make up a preventive maintenance task, from its definition to its completion. Because most preventive maintenance tasks are commonly performed at scheduled intervals, parts of the preventive maintenance cycle repeat, based on those intervals.
<b>preventive maintenance schedule</b>	The combination of service types that apply to a specific piece of equipment, as well as the intervals at which each service type is scheduled to be performed.
<b>primary service type</b>	A service type to which you can link related service types. For example, for a particular piece of equipment, you might set up a primary service type for a 1000-hour inspection and a linked service type for a 500-hour inspection. The 1000-hour inspection includes all of the tasks performed at 500 hours. When a primary service type is scheduled to be performed, the system schedules the linked service type. See also linked service type.
<b>pristine environment</b>	An EnterpriseOne environment that is used to test unaltered objects with PeopleSoft demonstration data or for training classes. You must have this environment so you can compare pristine objects that you modify.
<b>processing option</b>	A data structure that allows users to supply parameters that regulate the execution of a batch program or report.
<b>product data management (PDM)</b>	In PeopleSoft EnterpriseOne software, the system that enables a business to organize and maintain information about each item which it manufactures. Features of this system, such as bills of material, work centers, and routings, define the relationships among parents and components, and how they can be combined to manufacture an item. PDM also provides data for other manufacturing systems including Manufacturing Accounting, Shop Floor Management, and Manufacturing and Distribution Planning.
<b>product line</b>	A group of products with similarity in manufacturing procedures, marketing characteristics, or specifications that allow them to be aggregated for planning; marketing; and, occasionally, costing.
<b>product/process definition</b>	A combination of bill of material (recipe, formula, or both) and routing (process list). Organized into tasks with a statement of required consumed resources and produced resources.
<b>production environment</b>	An EnterpriseOne environment in which users operate EnterpriseOne software.
<b>program temporary</b>	A representation of changes to PeopleSoft software that your organization

<b>fix (PTF)</b>	receives on magnetic tapes or diskettes.
<b>project</b>	[In EnterpriseOne] A virtual container for objects being developed in Object Management Workbench.
<b>projected cost</b>	The target expenditure in added value for material, labor, and so on, during manufacture. See also standard cost.
<b>promotion path</b>	The designated path for advancing objects or projects in a workflow.
<b>protocollo</b>	See registration number.
<b>PST</b>	Provincial sales tax. A tax that is assessed by individual provinces in Canada.
<b>published table</b>	Also called a “Master” table, this is the central copy to be replicated to other machines and resides on the “publisher” machine. The Data Replication Publisher Table (F98DRPUB) identifies all of the published tables and their associated publishers in the enterprise.
<b>publisher</b>	The server that is responsible for the published table. The Data Replication Publisher Table (F98DRPUB) identifies all of the published tables and their associated publishers in the enterprise.
<b>pull replication</b>	One of the EnterpriseOne methods for replicating data to individual workstations. Such machines are set up as pull subscribers that use EnterpriseOne’s data replication tools. The only time that pull subscribers are notified of changes, updates, and deletions is when they request such information. The request is in the form of a message that is sent, usually at startup, from the pull subscriber to the server machine that stores the Data Replication Pending Change Notification table (F98DRPCN).
<b>query by example (QBE)</b>	Located at the top of a detail area, this area is used to search for data to display in the detail area.
<b>rate scheduling</b>	A method of scheduling product or manufacturing families, or both.  Also a technique to determine run times and quantities of each item within the family to produce enough of each individual product to satisfy demand until the family can be scheduled again.
<b>rate type</b>	For currency exchange transactions, the rate type distinguishes different types of exchange rates. For example, you can use both period average and period-end rates, distinguishing them by rate type.
<b>real-time</b>	Pertaining to information processing that returns a result so rapidly that the interaction appears to be instantaneous.
<b>receipt routing</b>	A series of steps that is used to track and move items within the receipt process. The steps might include in-transit, dock, staging area, inspection, and stock.
<b>referential integrity</b>	Ensures that a parent record cannot be deleted from the database when a child record for exists.

<b>regenerable</b>	Source code for EnterpriseOne business functions can be regenerated from specifications (business function names). Regeneration occurs whenever an application is recompiled, either for a new platform or when new functionality is added.
<b>register types and classes</b>	In Italian VAT Summary Reporting, the classification of VAT transactions.
<b>relationship</b>	Links tables together and facilitates joining business views for use in an application or report. Relationships that are created are based on indexes.
<b>rélevé d'identité bancaire (RIB)</b>	In France, the term that indicates the bank transit code, account number, and check digit that are used to validate the bank transit code and account number. The bank transit code consists of the bank code and agency code. The account number is alphanumeric and can be as many as 11 characters. PeopleSoft supplies a validation routine to ensure RIB key correctness.
<b>remessa</b>	In Brazil, the remit process for A/R.
<b>render</b>	To include external data in displayed content through a linking mechanism.
<b>repassé</b>	In Brazil, a discount of the ICMS tax for interstate transactions. It is the adjustment between the interstate and the intrastate ICMS tax rates.
<b>replenishment point</b>	The location on or near the production line where additional components or subassemblies are to be delivered.
<b>replication server</b>	A server that is responsible for replicating central objects to client machines.
<b>report design aid (RDA)</b>	The EnterpriseOne GUI tool for operating, modifying, and copying report batch applications.
<b>repost</b>	In Sales, the process of clearing all commitments from locations and restoring commitments, based on quantities from the Sales Order Detail table (F4211).
<b>resident</b>	Pertaining to computer programs or data while they remain on a particular storage device.
<b>retorno</b>	In Brazil, the receipt process for A/R.
<b>RIB</b>	See rélevé d'identité bancaire.
<b>ricevute bancarie (RiBa)</b>	In Italy, the term for accounts receivable drafts.
<b>riepilogo IVA</b>	Summary VAT monthly report. In Italy, the term for the report that shows the total amount of VAT credit and debit.
<b>ritenuta d'acconto</b>	In Italy, the term for standard withholding tax.
<b>rollback</b>	[In database management] A feature or command that undoes changes in database transactions of one or more records.
<b>rollup</b>	See cost rollup.

<b>row exit</b>	[In EnterpriseOne] An application shortcut, available as a button on the Row Exit bar or as a menu selection, that allows users to open a form that is related to the highlighted grid record.
<b>runtime</b>	The period of time when a program or process is running.
<b>SAD</b>	The German name for a Swiss payment format that is accepted by Postfinance.
<b>SAR</b>	See software action request.
<b>scalability</b>	The ability of software, architecture, hardware, or a network to support software as it grows in size or resource requirements.
<b>scripts</b>	A collection of SQL statements that perform a specific task.
<b>scrub</b>	To remove unnecessary or unwanted characters from a string.
<b>search/select</b>	A type of form that is used to search for a value and return it to the calling field.
<b>selection</b>	Found on PeopleSoft menus, selections represent functions that you can access from a menu. To make a selection, type the associated number in the Selection field and press Enter.
<b>serialize</b>	To convert a software object into a stream of bytes to store on a disk or transfer across a network.
<b>server map</b>	The server view of the object configuration mapping.
<b>server workbench</b>	During the Installation Workbench process, Server Workbench copies the server configuration files from the Planner data source to the System release number data source. It also updates the Server Plan detail record to reflect completion.
<b>service interval</b>	The frequency at which a service type is to be performed. Service intervals can be based on dates, periods, or statistical units that are user defined. Examples of statistical units are hours, miles, and fuel consumption.
<b>service type</b>	An individual preventive maintenance task or procedure, such as an inspection, lubrication, or overhaul. Service types can apply to a specific piece of equipment or to a class of equipment. You can specify that service types come due based on a predetermined service interval, or whenever the task that is represented by the service type becomes necessary.
<b>servlet</b>	A [small] program that extends the functionality of a Web server by generating dynamic content and interacting with Web clients by using a request-response paradigm.
<b>share path</b>	The network node under which one or more servers or objects reside.
<b>shop floor management</b>	A system that uses data from multiple system codes to help develop, execute, and manage work orders and rate schedules in the enterprise.
<b>silent mode</b>	A method for installing or running a program that does not require any user intervention.

<b>silent post</b>	A type of post that occurs in the background without the knowledge of the user.
<b>simulated cost</b>	After a cost rollup, the cost of an item, operation, or process according to the current cost scenario. This cost can be finalized by running the frozen update program. You can create simulated costs for a number of cost methods—for example, standard, future, and simulated current costs. See also cost rollup.
<b>single-byte character set (SBCS)</b>	An encoding scheme in which each alphabetic character is represented by one byte. Most Western languages, such as English, can be represented by using a single-byte character set.
<b>single-level tracking</b>	Finding all immediate parents where a specific lot has been used (consumed).
<b>single-voyage (spot) charter</b>	An agreement for a single voyage between two ports. The payment is made on the basis of tons of product delivered. The owner of the vessel is responsible for all expenses.
<b>slimer</b>	A script that changes data in a table directly without going through a regular database interface.
<b>smart field</b>	A data dictionary item with an attached business function for use in the Report Design Aid application.
<b>SOC</b>	The Italian term for a Swiss payment format that is accepted by Postfinance.
<b>soft commitment</b>	The number of items that is reserved for sales orders or work orders in the primary units of measure.
<b>soft error</b>	An error from which an operating system or program is able to recover.
<b>software action request (SAR)</b>	An entry in the AS/400 database that is used for requesting modifications to PeopleSoft software.
<b>SOG</b>	The French term for a Swiss payment format that is accepted by Postfinance.
<b>source directory</b>	The path code to the business function source files belonging to the shared library that is created on the enterprise server.
<b>special period/year</b>	The date that determines the source balances for an allocation.
<b>specification merge</b>	The Specification merge is comprised of three merges: Object Librarian merge (via the Object Management Workbench). Versions List merge. Central Objects merge. The merges blend customer modifications with data that accompanies a new release.
<b>specification table merge workbench</b>	During the Installation Workbench process, Specification Table Merge Workbench runs the batch applications that update the specification tables.

<b>specifications</b>	A complete description of an EnterpriseOne object. Each object has its own specification, or name, which is used to build applications.
<b>spot charter</b>	See single-voyage charter.
<b>spot rates</b>	An exchange rate that is entered at the transaction level. Spot rates are not used on transactions between two EMU member currencies because exchange rates are irrevocably fixed to the euro.
<b>stamp tax</b>	In Japan, a tax that is imposed on drafts payable, receipts over 30000 Japanese yen, and all contracts. The party that issues any of the above documents is responsible for this tax.
<b>standalone</b>	Operating or capable of operating independently of certain other components of a computer system.
<b>standard cost</b>	The expected, or target cost of an item, operation, or process. Standard costs represent only one cost method in the Product Costing system. You can also calculate, for example, future costs or current costs. However, the Manufacturing Accounting system uses only standard frozen costs.
<b>standard costing</b>	A costing method that uses cost units that are determined before production. For management control purposes, the system compares standard costs to actual costs and computes variances.
<b>subprocess</b>	A process that is triggered by and is part of a larger process, and that generally consists of activities.
<b>subscriber table</b>	The Subscriber table (F98DRSUB), which is stored on the Publisher Server with the Data Replication Publisher table (F98DRPUB), that identifies all of the subscriber machines for each published table.
<b>summary</b>	The presentation of data or information in a cumulative or totaled manner in which most of the details have been removed. Many systems offer forms and reports that summarize information which is stored in certain tables. Contrast with detail.
<b>super backflush</b>	To create backflush transactions for material, labor, or both, against a work order at predefined pay points in the routing. By doing so, you can relieve inventory and account for labor amounts at strategic points throughout the manufacturing process.
<b>supersession</b>	Specification that a new product is replacing an active product on a specified effective date.
<b>supplemental data</b>	Additional types of data for customers and suppliers. You can enter supplemental data for information such as notes, comments, plans, or other information that you want in a customer or supplier record. The system maintains this data in generic databases, separate from the standard master tables (Customer Master, Supplier Master, and Address Book Master).

<b>supplying location</b>	The location from which inventory is transferred once quantities of the item on the production line have been depleted. In kanban processing, the supplying location is the inventory location from which materials are transferred to the consuming location when the containers are replenished.
<b>system code</b>	A numeric or alphanumeric designation that identifies a specific system in EnterpriseOne software.
<b>system function</b>	[In EnterpriseOne] A named set of pre-packaged, re-usable instructions that can be called from event rules.
<b>table access management (TAM)</b>	The EnterpriseOne component that handles the storage and retrieval of user defined data. TAM stores information such as data dictionary definitions; application and report specifications; event rules; table definitions; business function input parameters and library information; and data structure definitions for running applications, reports, and business functions.
<b>table conversion workbench</b>	During the Installation Workbench process, Table Conversion Workbench runs the table conversions that change the technical and application tables to the format for the new release of EnterpriseOne. It also updates the Table Conversions and Controls detail records to reflect completion.
<b>table design aid (TDA)</b>	An EnterpriseOne GUI tool for creating, modifying, copying, and printing database tables.
<b>table event rules</b>	Use table event rules to attach database triggers (or programs) that automatically run whenever an action occurs against the table. An action against a table is referred to as an event. When you create an EnterpriseOne database trigger, you must first determine which event will activate the trigger. Then, use Event Rules Design to create the trigger. Although EnterpriseOne allows event rules to be attached to application events, this functionality is application-specific. Table event rules provide embedded logic at the table level.
<b>table handle</b>	A pointer into a table that indicates a particular row.
<b>table space</b>	[In relational database management systems] An abstract collection of containers in which database objects are stored.
<b>task</b>	[In Solution Explorer and EnterpriseOne Menu] A user defined object that can initiate an activity, process, or procedure.
<b>task view</b>	A group of tasks in Solution Explorer or EnterpriseOne Menu that are arranged in a tree structure.
<b>termo de abertura</b>	In Brazil, opening terms for the transaction journal.
<b>termo de encerramento</b>	In Brazil, closing terms for the transaction journal.
<b>three-tier processing</b>	The task of entering, reviewing, approving, and posting batches of transactions.
<b>three-way voucher match</b>	The process of comparing receipt information to supplier's invoices to create vouchers. In a three-way match, you use the receipt records, the purchase order, and the invoice to create vouchers.

<b>threshold percentage</b>	In Capital Asset Management, the percentage of a service interval that you define as the trigger for maintenance to be scheduled. For example, you might set up a service type to be scheduled every 100 hours with a threshold percentage of 90 percent. When the equipment accumulates 90 hours, the system schedules the maintenance.
<b>throughput agreement</b>	A service agreement in which a business partner agrees to store and manage product for another business partner for a specified time period. The second partner actually owns the stock that is stored in the first partner's depot, although the first partner monitors the stock level; suggests replenishments; and unloads, stores, and delivers product to the partner or its customers. The first partner charges a fee for storing and managing the product.
<b>throughput reconciliation</b>	Reconcile confirmed sales figures in a given period with the measured throughput, based on the meter readings. This process is designed to catch discrepancies that are due to transactions not being entered, theft, faulty meters, or some combination of these factors. This reconciliation is the first stage. See also operational reconciliation.
<b>token</b>	[In Object Management Workbench] A flag that is associated with each object which indicates whether you can check out the object.
<b>tolerance range</b>	The amount by which the taxes that you enter manually can vary from the tax that is calculated by the system.
<b>TP monitor</b>	Transaction Processing monitor. A monitor that controls data transfer between local and remote terminals and the applications that originated them. TP monitors also protect data integrity in the distributed environment and can include programs that validate data and format terminal screens.
<b>tracing</b>	The act of researching a lot by going backward, to discover its origin.
<b>tracking</b>	The act of researching a lot by going forward, to discover where it is used.
<b>transaction set</b>	An electronic business transaction (EDI Standard document) composed of segments.
<b>transclude</b>	To include the external data in the displayed content through a linking mechanism.
<b>transfer order</b>	An order that is used to ship inventory between branch/plants within your company and to maintain an accurate on-hand inventory amount. An interbranch transfer order creates a purchase order for the shipping location and a sales order for the receiving location.
<b>translation adjustment account</b>	An optional G/L account used in currency balance restatement to record the total adjustments at a company level.
<b>translator software</b>	The software that converts data from an application table format to an EDI Standard Format, and from EDI Standard Format to application table format. The data is exchanged in an EDI Standard, such as ANSI ASC X12, EDIFACT, UCS, or WINS.

<b>tree structure</b>	A type of graphical user interface that displays objects in a hierarchy.
<b>trigger</b>	Allows you to attach default processing to a data item in the data dictionary. When that data item is used on an application or report, the trigger is invoked by an event which is associated with the data item. EnterpriseOne also has three visual assist triggers:  Calculator.  Calendar.  Search form.
<b>two-way voucher match</b>	The process of comparing purchase order detail lines to the suppliers' invoices to create vouchers. You do not record receipt information.
<b>universal batch engine (UBE)</b>	[In EnterpriseOne] A type of application that runs a noninteractive process.
<b>unnormalized</b>	Data that is a random collection of data elements with repeating record groups scattered throughout. Also see Normalized.
<b>user overrides merge</b>	The User Overrides merge adds new user override records into a customer's user override table.
<b>user-defined code (UDC)</b>	A value that a user has assigned as being a valid entry for a given or specific field.
<b>utility</b>	A small program that provides an addition to the capabilities which are provided by an operating system.
<b>variable numerator allocations</b>	A procedure that allocates or distributes expenses, budgets, adjustments, and so on, among business units, based on a variable.
<b>variable quantity</b>	A term that indicates the bill of material relationship between a parent item and its components or ingredients. When a bill of material component has a variable quantity relationship to its parent, the amount of the component changes when the software calculates parts list requirements for different work order quantities. Contrast with fixed quantity.
<b>variance</b>	1. In Product Costing and Manufacturing Accounting, the difference between the frozen standard cost, the current cost, the planned cost, and the actual cost. For example, the difference between the frozen standard cost and the current cost is an engineering variance. Frozen standard costs come from the Cost Components table, and the current costs are calculated by using the current bill of material, routing, and overhead rates.  2. In Capital Asset Management, the difference between revenue that is generated by a piece of equipment and costs that are incurred by the equipment.
<b>versions list merge</b>	The Versions List merge preserves any non-XJDE and non-ZJDE version specifications for objects that are valid in the new release as well as their processing options data.
<b>VESR</b>	Verfahren Einzahlungsschein mit Referenznummer. The processing of an ESR pay slip with reference line through accounts receivable and accounts payable.

<b>visual assist</b>	Forms that can be invoked from a control to assist the user in determining what data belongs in the control.
<b>voucher logging</b>	The process of entering vouchers without distributing amounts to specific G/L accounts. The system initially distributes the total amount of each voucher to a G/L suspense account, where it is held until you redistribute it to the correct G/L account.
<b>wareki date format</b>	In Japan, a calendar format, such as Showa or Heisei. When a new emperor begins to reign, the government chooses the title of the date format and the year starts over at one. For instance, January 1, 1998, is equal to Heisei 10, January 1st.
<b>wash down</b>	A minor cleanup between similar product runs. Sometimes used in reference to the sanitation process of a food plant.
<b>wchar_t</b>	An internal type of a wide character. Used for writing portable programs for international markets.
<b>web server</b>	A server that sends information as requested by a browser and uses the TCP/IP set of protocols.
<b>work order life cycle</b>	In Capital Asset Management, the sequence of events through which a work order must pass to accurately communicate the progress of the maintenance tasks that it represents.
<b>workfile</b>	A system-generated file that is used for temporary data processing.
<b>workflow</b>	According to the Workflow Management Coalition, workflow means “the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.”
<b>workgroup server</b>	A network server usually containing subsets of data that are replicated from a master network server.
<b>WorldSoftware architecture</b>	The broad spectrum of application design and programming technology that PeopleSoft uses to achieve uniformity, consistency, and complete integration throughout its software.
<b>write payment</b>	A step in processing payments. Writing payments includes printing checks, drafts, and creating a bank tape table.
<b>write-off</b>	A method for getting rid of inconsequential differences between amounts. For example, you can apply a receipt to an invoice and write off the difference. You can write off both overpayments and underpayments.

<b>Z file</b>	For store and forward (network disconnected) user, EnterpriseOne store-and-forward applications perform edits on static data and other critical information that must be valid to process an order. After the initial edits are complete, EnterpriseOne stores the transactions in work tables on the workstation. These work table are called Z files. When a network connection is established, Z files are uploaded to the enterprise server; and the transactions are edited again by a master business function. The master business function then updates the records in your transaction files.
<b>z-process</b>	A process that converts inbound data from an external system into an EnterpriseOne software table or converts outbound data into an interface table for an external system to access.
<b>zusammenfassende melding</b>	In Germany, the term for the EU Sales Listing.



# Index

## A

- Abbreviations in message column of event stream event pane, 13
- Action tools in Virtual Runner, 31
- Adding loops, 36
- All Virtual Scripts list box, 29
- API playback timing, 47
- Automatic parameter value linking, 18
- AutoPilot, 4, 5
  - script creation, 5
  - script playback, 5
  - script playback configuration, 5
- AutoPilot and EnterpriseOne software code, 5
- AutoPilot playback configuration, 5
- AutoPilot Playback Results Detail Table (F97214), 4
- AutoPilot script creation, 5
- AutoPilot script playback, 5

## B

- Buttons in the event stream event pane, 13
- Buttons in the script list pane, 15

## C

- Capturing and importing test results, 32
  - capturing results, 33
  - importing test results, 34
  - viewing test results, 34
- Capturing test results, 33
- Code for data capture, 4
- Columns in the script list pane, 15
- Command line for Virtual Script Player launch, 26
- Command section of initialization file parameters, 23
- Correcting uninitialized user handle errors, 67
- Creating master scripts, 41
- Creating virtual scripts, 32
  - capturing and importing test results, 32
    - capturing test results, 33
    - importing test results, 34
    - viewing test results, 34
  - editing the virtual script, 35
    - adding loops, 36
    - creating master scripts, 41
    - linking values in entry scripts, 39

- linking values in inquiry scripts, 39
- script generation, 40
- value-linking parameters, 37

## D

- Data capture components, 4
  - AutoPilot, 5
    - script creation, 5
    - script playback, 5
    - script playback configuration, 5
  - AutoPilot and EnterpriseOne software code, 5
  - AutoPilot Playback Results Detail Table (F97214), 9
    - event stream, 8
    - functions of AutoPilot hooks, 6
    - functions of EnterpriseOne software hooks, 6
- Data capture for virtual scripts, 4
  - AutoPilot, 4
  - AutoPilot Playback Results Detail Table (F97214), 4
  - code for data capture, 4
  - data capture components, 4
    - AutoPilot, 5
    - AutoPilot and EnterpriseOne software code, 5
    - AutoPilot Playback Results Detail Table (F97214), 9
    - event stream, 8
  - event stream, 4
- Date formatting, 28
- Debugging LoadRunner, 67
- Debugging Virtual Runner, 66
- Debugging Virtual User Tool scripts, 59
  - correcting uninitialized user handle errors, 67
  - diagnosing business function failures in EnterpriseOne, 60
  - displaying business function parameters, 59
  - identifying and correcting duplicate key errors, 64
  - preventing multiple script playback problems, 66
    - debugging LoadRunner, 67
    - debugging Virtual Runner, 66
  - rectifying irregular transaction times, 65
  - troubleshooting value-linking errors, 61
    - identifying and correcting variable value-linking errors, 63

- researching value-linking errors in the Virtual Script Editor, 61
  - verifying that value linking is functioning, 62
- verifying the validity of Virtual User Tool script data, 64
- Defining a script using LoadRunner, 45
- Defining the host machine using LoadRunner, 45
- Defining virtual users using LoadRunner, 45
- Diagnosing a OneWorld environment problem, 57
- Diagnosing business function failures in EnterpriseOne, 60
- Displaying business function parameters, 59
- Dynamic loop creation, 19
- Dynamic loop designation, 19
- Dynamic loop editing, 20

## E

- Editing the virtual script, 35
  - adding loops, 36
  - creating master scripts, 41
  - linking values in entry scripts, 39
  - linking values in inquiry scripts, 39
  - script generation, 40
  - value-linking parameters, 37
- EnterpriseOne environment initialization by the Virtual Script Player, 27
- EnterpriseOne Virtual User Tool components, 10
  - Virtual Script Editor, 11
  - Virtual Script Player, 23
- creating virtual scripts
  - capturing and importing test results, 32
- data capture for virtual scripts, 4
  - AutoPilot, 4
  - AutoPilot Playback Results Detail Table (F97214), 4
  - code for data capture, 4
  - data capture components, 4
  - event stream, 4
- special considerations for simulated playback, 47
  - synchronous and asynchronous calls, 52
  - think times, 53
- Virtual Script Editor
  - parameter value linking, 16
- EnterpriseOne Virtual User Tool components, 10
  - Virtual Script Editor, 11
    - parameter detail pane, 14

- script list pane, 15
  - thread identification, 20
  - virtual script generation, 22
- Virtual Script Player, 23
  - command line launch, 26
  - initialization file parameters, 23
- Event graph, 14
- Event pane, 12
- Event stream, 4, 8
- Event stream components
  - event pane
    - abbreviations in message column, 13
- Event stream event pane
  - abbreviations in message column, 13
  - buttons, 13

## F

- Finding error entries in the Virtual User Tool log file, 55
- Functions of AutoPilot hooks, 6
- Functions of EnterpriseOne software hooks, 6

## G

- Gathering LoadRunner results, 46
- Generating the virtual script, 40

## H

- hRequest handle value linking, 20

## I

- Identifying a OneWorld environment problem, 57
  - diagnosing a OneWorld environment problem, 57
  - investigating OneWorld errors, 58
- Identifying and correcting duplicate key errors, 64
- Identifying and correcting variable value-linking errors, 63
- Importing test results, 34
- Initialization file parameters for Virtual Script Player, 23
  - Command section, 23
  - Log section, 25
  - Paths section, 24
  - Timing section, 24
- Interthread timing, 49
- Investigating OneWorld errors, 58

## L

- Launching and managing multiple script playback, 45
  - defining a script, 45
  - defining the host machine, 45
  - defining virtual users, 45
  - gathering LoadRunner results, 46
  - running virtual playback from the LoadRunner controller, 46
  - setting rendezvous points, 46
- Limitations of Virtual Script Player, 28
- Linking parameters in entry scripts, 39
- Linking parameters in inquiry scripts, 39
- Locating the causes of Virtual User Tool script failure
  - finding error entries in the log file, 55
  - identifying a OneWorld environment problem, 57
  - locating the log file in the event of early script failure, 56
  - setting the MessageLevel parameter, 56
- Locating the causes of Virtual User Tool script failures, 54
  - identifying a OneWorld environment problem
    - diagnosing a OneWorld environment problem, 57
    - investigating OneWorld errors, 58
- Locating the log file in the event of early script failure, 56
- Log section of initialization file parameters, 25
- Looping identification, 18
  - dynamic loop creation, 19
  - dynamic loop designation, 19
  - dynamic loop editing, 20

## M

- Manual parameter value linking, 18
- Master Scripts list box, 29
- Modes of operation for Virtual Script Player, 27

## O

- OneWorld Virtual User Tool components
  - Virtual Runner, 30
  - VSMEditor, 29
- creating virtual scripts, 32
  - editing the virtual script, 35
- running virtual scripts, 42
  - from a command line, 42
  - from a single workstation, 42

- launching and managing multiple script playback, 45
  - using Virtual Runner, 43
- special considerations for simulated playback call level, 51
  - playback timing, 47
- OneWorld Virtual User Tool components
  - Virtual Runner, 30
  - Virtual Script Editor
    - event graph, 14
    - event pane, 12
    - hRequest handle value linking, 20
    - looping identification, 18
    - timing interval maintenance, 21
  - Virtual Script Player
    - date formatting, 28
    - limitations, 28
    - modes of operation, 27
    - preprocessing of valid values list data, 27
    - script failure, 28
    - valid values list processing, 27
  - VSMEditor, 29
    - All Virtual Scripts list box, 29
    - Master Scripts list box, 29
    - VSM files, 30

## P

- Parameter detail pane, 14
- Parameter identification
  - source parameters, 17
  - target parameters, 17
- Parameter value linking, 16
  - automatic, 18
  - manual, 18
- Paths section of initialization file parameters, 24
- Placement of hooks for data capture, 5
- Playback timing, 47
  - API playback timing, 47
  - interthread timing, 49
- Player session columns in Virtual Runner, 30
- Preprocessing of valid values list data, 27
- Preventing multiple script playback problems, 66
  - debugging LoadRunner, 67
  - debugging Virtual Runner, 66

## R

- Rectifying irregular transaction times, 65
- Researching value-linking errors in the Virtual Script Editor, 61

- Running virtual playback from the LoadRunner controller, 46
- Running virtual scripts, 42
  - from a command line, 42
  - from a single workstation, 42
  - launching and managing multiple script playback, 45
    - defining a script, 45
    - defining the host machine, 45
    - defining virtual users, 45
    - gathering LoadRunner results, 46
    - running virtual playback from the LoadRunner controller, 46
    - setting rendezvous points, 46
    - using Virtual Runner, 43
- Running virtual scripts from a command line, 42
- Running virtual scripts from a single workstation, 42
- Running virtual scripts using Virtual Runner, 43

## S

- Script failure, 28
- Script list pane, 15
  - buttons, 15
  - columns, 15
- Setting rendezvous points using LoadRunner, 46
- Setting the MessageLevel parameter, 56
- Source parameters, 17
- Special considerations for simulated playback, 47
  - call level, 51
  - playback timing, 47
    - API playback timing, 47
    - interthread timing, 49
  - synchronous and asynchronous calls, 52
  - think times, 53
- Synchronous and asynchronous calls, 52

## T

- Tables
  - AutoPilot Playback Results Detail Table (F97214), 9
- Target parameters, 17
- Think times, 53
- Thread identification, 20
- Timing interval maintenance, 21
- Timing section of initialization file parameters, 24
- Troubleshooting tips and techniques for Virtual User Tool scripts, 54
  - debugging Virtual User Tool scripts, 59

- correcting uninitialized user handle errors, 67
- diagnosing business function failures in EnterpriseOne, 60
- displaying business function parameters, 59
- identifying and correcting duplicate key errors, 64
- preventing multiple script playback problems, 66
- rectifying irregular transaction times, 65
- troubleshooting value-linking errors, 61
- verifying the validity of Virtual User Tool script data, 64
- locating the causes of Virtual User Tool script failures, 54
  - finding error entries in the log file, 55
  - identifying a OneWorld environment problem, 57
  - locating the log file in the event of early script failure, 56
  - setting the MessageLevel parameter, 56
- Troubleshooting value-linking errors, 61
  - identifying and correcting variable value-linking errors, 63
  - researching value-linking errors in the Virtual Script Editor, 61
  - verifying that value linking is functioning, 62

## V

- Valid values list processing, 27
- Value-linking parameters, 37
- Verifying that value linking is functioning, 62
- Verifying the validity of Virtual User Tool script data, 64
- Viewing test results, 34
- Virtual Runner, 30
  - components
    - Actions tools, 31
    - Player session columns, 30
- Virtual script editor
  - parameter identification
    - source parameters, 17
    - target parameters, 17
- Virtual Script Editor
  - event graph, 14
  - event pane, 12
  - hRequest handle value linking, 20
  - looping identification, 18
    - dynamic loop creation, 19
    - dynamic loop designation, 19
    - dynamic loop editing, 20

- parameter detail pane, 14
- parameter value linking, 16
  - automatic, 18
  - manual, 18
- script list pane, 15
- thread identification, 20
- timing interval maintenance, 21
- virtual script generation, 22
- Virtual script generation, 22
- Virtual Script Player, 23
  - command line launch, 26
  - date formatting, 28
  - EnterpriseOne environment initialization, 27
  - initialization file parameters, 23
  - Command section, 23
  - Log section, 25
  - Paths section, 24
  - Timing section, 24
- limitations, 28
- modes of operation, 27
- preprocessing of valid values list data, 27
- script failure, 28
- valid values list processing, 27
- VSM files, 30
- VSMEditor, 29
  - All Virtual Scripts list box, 29
  - Master Scripts list box, 29
  - VSM files, 30

