PeopleSoft

EnterpriseOne Virtual AutoPilot 8.9 PeopleBook

September 2003

EnterpriseOne
Virtual AutoPilot 8.9 PeopleBook
SKU REL9EVA0309

Table of Contents

OneWorld Virtual AutoPilot Overview

J.D. Edwards OneWorld Virtual AutoPilot (Virtual AutoPilot) is a collection of automated testing tools that you use to simulate on a single workstation the actions of more than one J.D. Edwards software user. You use the tool in conjunction with AutoPilot, an automated testing tool that allows you to create scripts that test J.D. Edwards software processes.

AutoPilot lets you capture and save results from each script playback session using hooks, in the form of code strategically placed both in AutoPilot and in J.D. Edwards software. AutoPilot's data capture and storage capability provides a precise record of a single playback session, including all API calls. Virtual AutoPilot enables you to use the data you capture during an AutoPilot playback session to generate scripts that you run to approximate actual stress on a server and on the network.

Virtual AutoPilot:

- Provides a manageable solution for the creation and playback of virtual scripts that test a system's scalability
- Captures and logs data about each script playback session
- Bypasses J.D. Edwards software user interface to run virtual scripts
- Tests applications regardless of Object Configuration Manager (OCM) settings and database caching
- Re-creates J.D. Edwards software's actual operations in a stressed server and network environment
- Supports *n-tier* testing
- Runs independently of most changes in J.D. Edwards software code
- Provides a mechanism for reporting and analyzing test results

In addition to being cost-efficient and resource-efficient, Virtual AutoPilot scripts are flexible because you can run them against any OCM mapping, operating system, or database.

Virtual AutoPilot batch-mode script testing offers the following additional benefits:

- Departments can reduce the amount of time required to create scripts.
- Companies can reduce the amount of time, personnel, and equipment invested in testing J.D. Edwards software applications.
- Performance engineers can determine the scalability of a system running J.D. Edwards software.
- Tools developers can test and debug jdekrnl and jdenet in an environment that realistically simulates heavy user load.
- Application developers can test OCM configurations and the performance of business functions in a stressed environment.
- Business partners can size hardware to meet the needs of customers using J.D. Edwards software.

The collection of tools that makes up Virtual AutoPilot is part of an automated testing tools architecture developed by J.D. Edwards. The following table presents the components of the architecture and summarizes the role of each in the process of creating virtual scripts:

Component of Virtual AutoPilot Architecture	Function in Virtual AutoPilot Script Creation
AutoPilot	Automated testing tool used to write scripts that test J.D. Edwards software applications by simulating user input.
Hooks in AutoPilot and in J.D. Edwards software	Code that captures and records all data generated during the playback of an AutoPilot script.
Event Stream	Record of the continuous series of events captured by AutoPilot during script playback. Resides within AutoPilot Playback Results Detail Table (F97214).
AutoPilot Playback Results Detail Table (F97214)	The J.D. Edwards software table in which the event stream is stored.
Virtual Script Editor	Tool that allows user to import an event stream and to create a Virtual AutoPilot script by applying rules that govern the way the script runs in batch mode.
Virtual AutoPilot script	The combination of events from the event stream and the rules inserted automatically by the Virtual Script Editor and inserted manually by the user.
Virtual Script Player	Tool that runs the Virtual AutoPilot script, simulating the work of the J.D. Edwards software run-time engine.
vap.log file	Text file that contains any error messages that the Virtual Script Player sends during playback.
Virtual Runner	Program that allows you to launch multiple Virtual AutoPilot scripts from a single workstation.

Each of these components is discrete. However, each plays a role that is integral to the entire process of creating a virtual script. We summarize the stages of the script creation process, the role of each component, and the way the roles relate as follows:

- Data capture

  You write an AutoPilot script and play it back. Hooks in AutoPilot and in J.D. Edwards software capture the event stream and store this record of events in AutoPilot Playback Results Detail Table (F97214).

- Virtual script creation

  You use the Virtual Script Editor to create the virtual script from the event stream.

- Virtual script playback

  After you generate the script, you use the Virtual Script Player to play the script. The Virtual Script Player assumes the role of the J.D. Edwards software run-time engine. Virtual AutoPilot bypasses J.D. Edwards software completely in playing back the virtual script.

- Error message generation

  During playback of a virtual script, the Virtual Script Player sends error messages to the vap.log file text file.

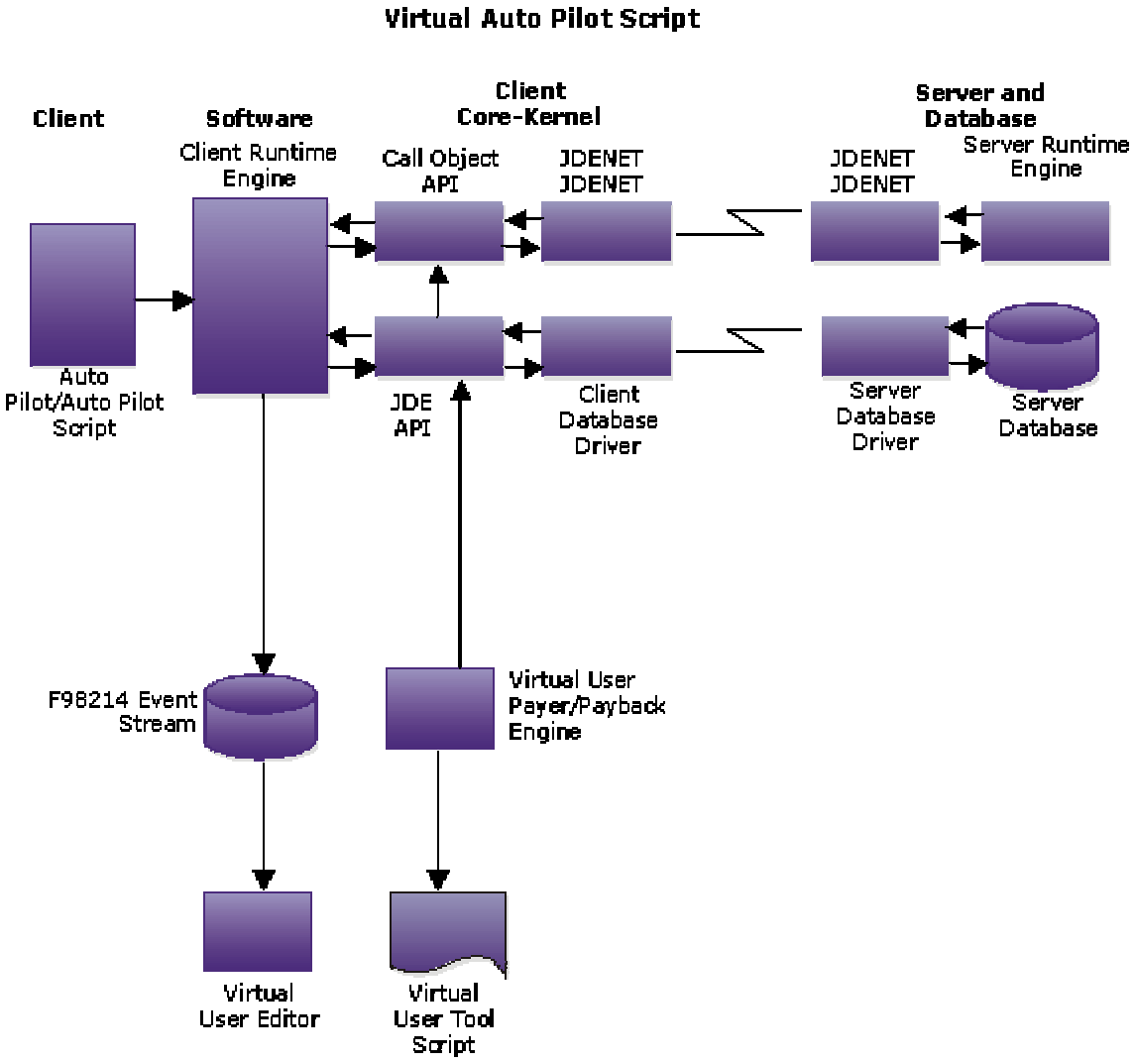
- Multiple script launch and playback analysis

  Virtual Runner and Mercury Interactive's LoadRunner load the testing tool to launch Virtual AutoPilot scripts and to save and analyze test results for each simulated user session. You use Virtual Runner to launch simulated multiple users on a single workstation; you use LoadRunner to launch multiple script playback sessions from more than one workstation.

  Note
  The number of simulated users is determined, in part, by the power of the workstations and the operating system.

Another way to conceptualize the Virtual AutoPilot architecture is to examine the flow of data from the time you write an AutoPilot script to the time you simulate multiple J.D. Edwards software sessions on a single workstation with the Virtual Script Player.

We also can view the Virtual AutoPilot architecture as it is positioned within the larger context of J.D. Edwards configurable network computing (CNC) architecture. Such a view shows the flow of data that occurs during the playback of an AutoPilot script. Note again that data captured in J.D. Edwards software and in AutoPilot during script playback is routed to AutoPilot Playback Results Detail Table (F97214). The Virtual Script Editor user imports the data and modifies it to produce a Virtual AutoPilot script that the Virtual Script Player runs to simulate J.D. Edwards software activity.



In summary, Virtual AutoPilot's design and architecture allow it to meet the challenge of accurately simulating server and network stress imposed by the dynamic interaction of multiple J.D. Edwards software users.

Data Capture for Virtual AutoPilot Scripts

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

The following components of J.D. Edwards automated testing tools architecture work together to capture, record, and store data about J.D. Edwards software processes, including the parameters of all API calls and all other J.D. Edwards software run-time events:

- AutoPilot, which allows you to write and play back a script to test J.D. Edwards software applications and to configure script playback so that AutoPilot captures and saves playback data
- Hooks, or code, that reside in J.D. Edwards 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 J.D. Edwards software event that occurs during script playback
- AutoPilot Playback Results Detail Table (F97214), which stores the event stream

The placement of J.D. Edwards software code is important to the creation of Virtual AutoPilot scripts. Because this code is positioned at the boundary between the J.D. Edwards software run-time engine and the J.D. Edwards 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 J.D. Edwards 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 J.D. Edwards 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:

AutoPilot	Allows you to write a script, play it back in J.D. Edwards software, and save the results of the playback.
Code that resides in J.D. Edwards software and in AutoPilot	Captures and records the data generated by AutoPilot and by J.D. Edwards software during script playback.
AutoPilot Playback Results Detail Table (F97214)	Stores the data generated during script playback as an event stream, which is a continuous record of every J.D. Edwards 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 J.D. Edwards software processes. Playing back an AutoPilot script simulates J.D. Edwards 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.

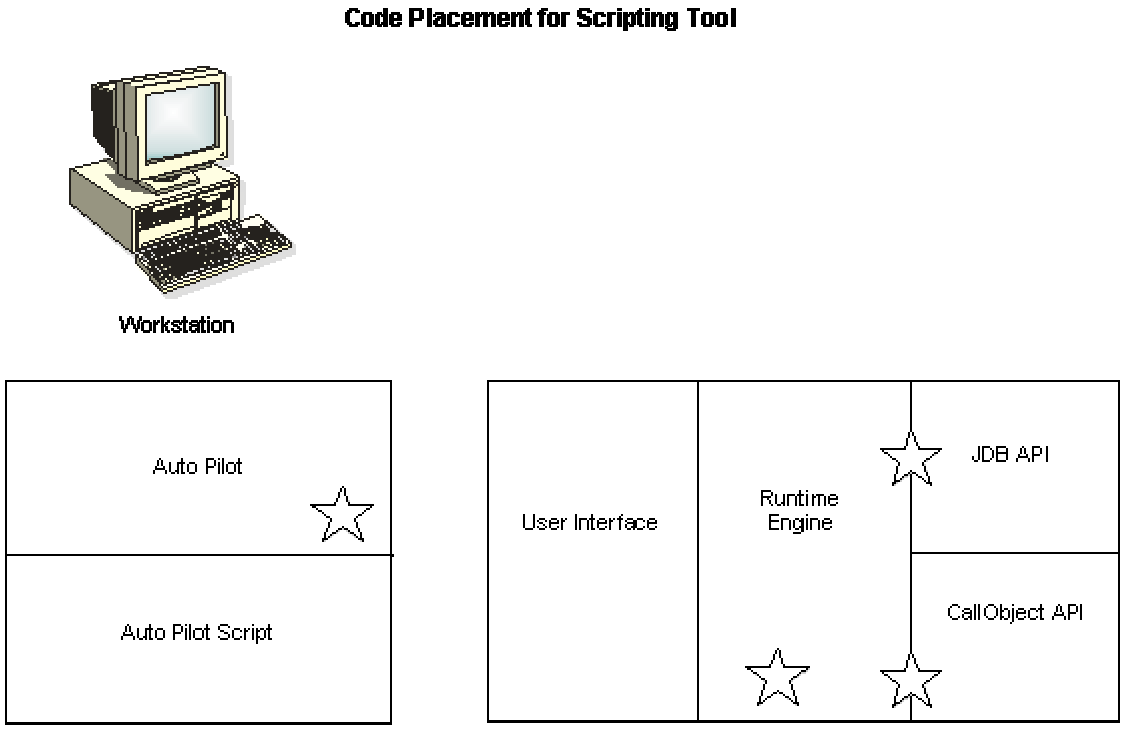
J.D. Edwards Software and AutoPilot Code

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

The placement of J.D. Edwards software code provides the following important advantages for creating OneWorld Virtual AutoPilot scripts:

- Comprehensive data capture, because code is positioned to capture both JDB and JDE CallObject API calls. This means that 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 J.D. Edwards software code relatively easy to handle.
- Flexibility in running scripts, because 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 J.D. Edwards software and AutoPilot code for the capture of data during playback of an AutoPilot script:



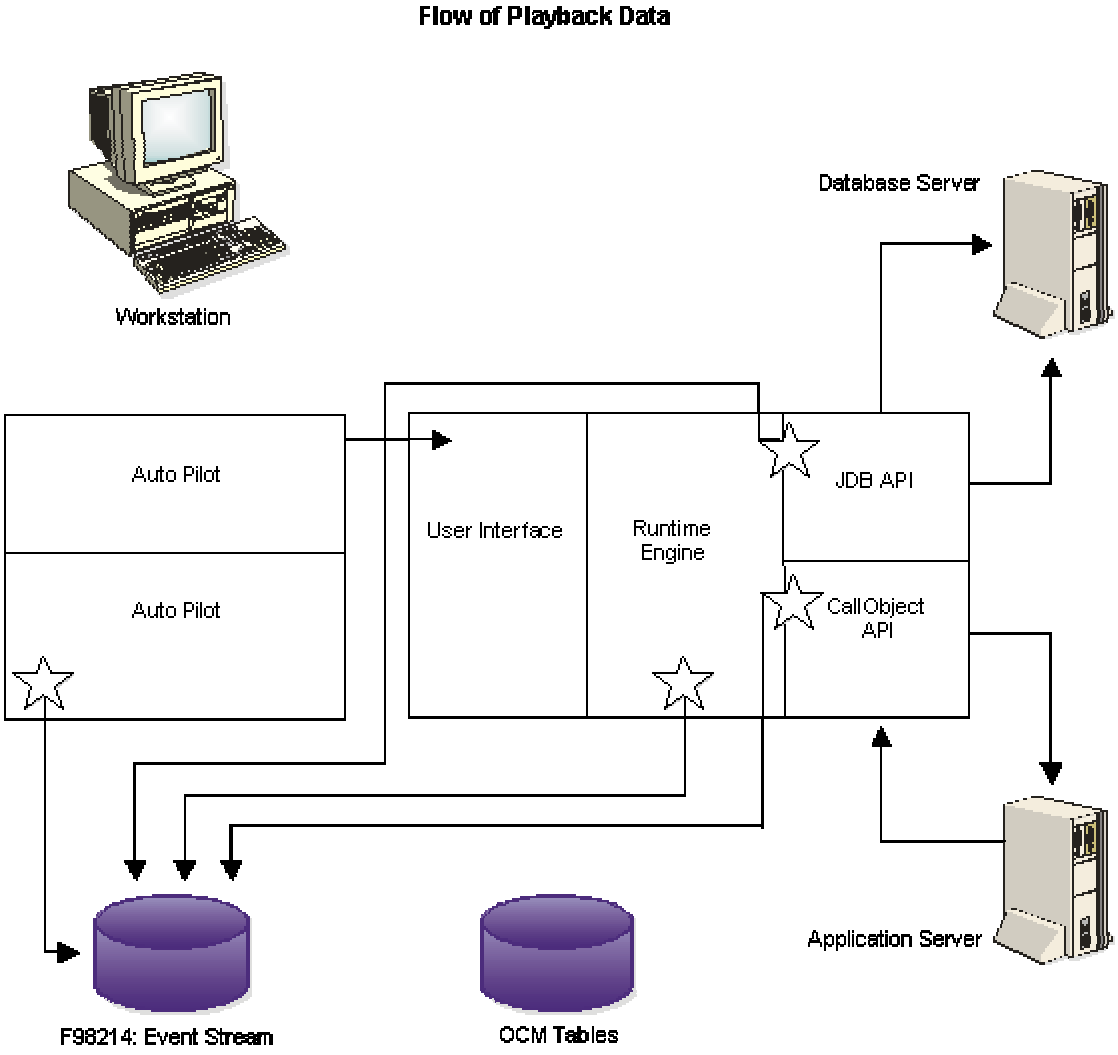
Code placed in J.D. Edwards software performs the following functions that lay the groundwork for the creation of a Virtual AutoPilot script:

- 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 J.D. Edwards 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 J.D. Edwards 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)

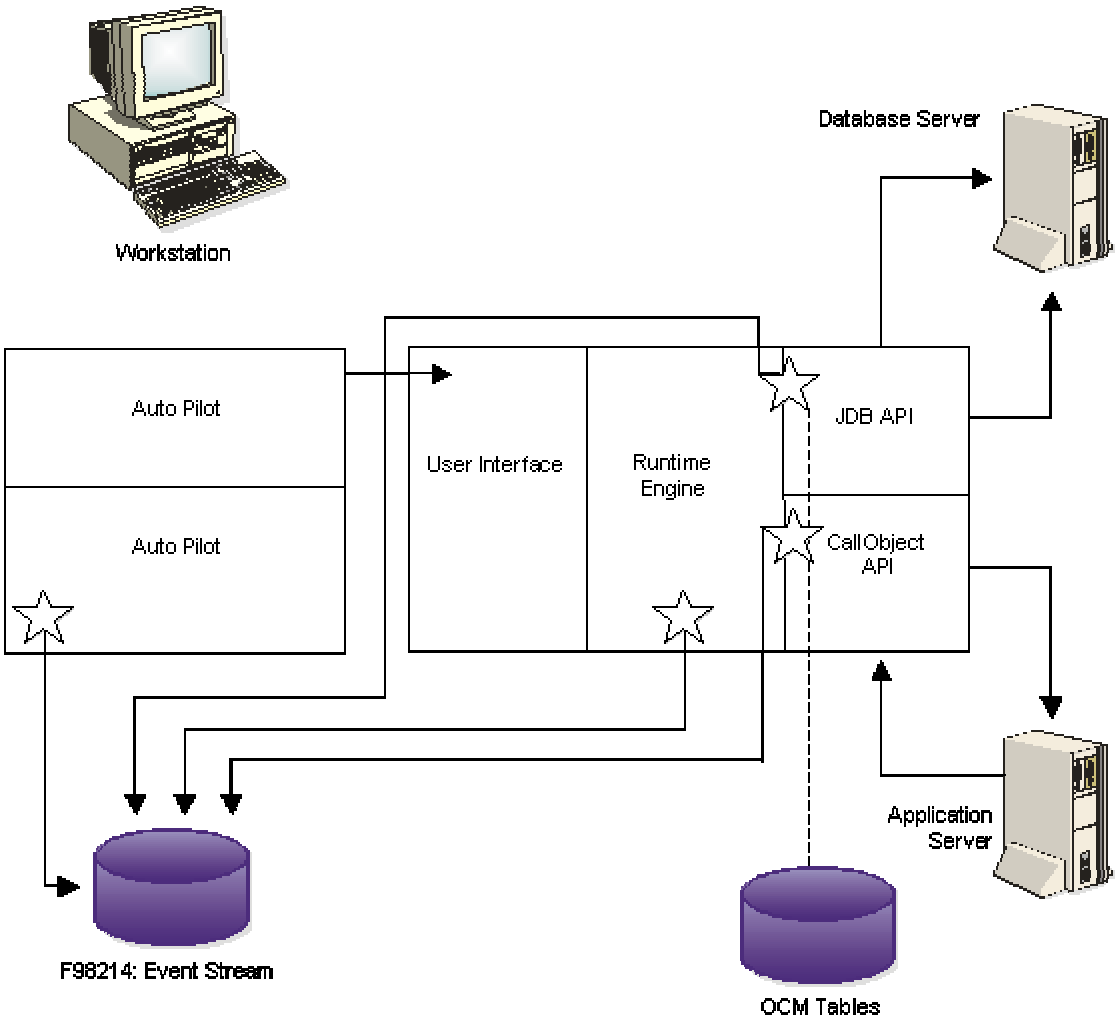
The following diagram illustrates the flow of playback data from the data-capture points to the AutoPilot Playback Results Detail Table (F97214), where the results are stored:



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 J.D. Edwards software code captures API call data before it reaches the OCM for mapping.

The dotted line in the following diagram illustrates the positioning of J.D. Edwards 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 J.D. Edwards software event that occurs during playback, including:

- JDB and CallObject API calls
- Thread identification
- Event rules
- J.D. Edwards software error and warning messages
- AutoPilot events confirming that the script and J.D. Edwards 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 OneWorld 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 OneWorld 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.

OneWorld Virtual AutoPilot Components

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

- AutoPilot
- Data-capture code in AutoPilot and in J.D. Edwards 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 OneWorld 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 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 run-time 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 J.D. Edwards 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

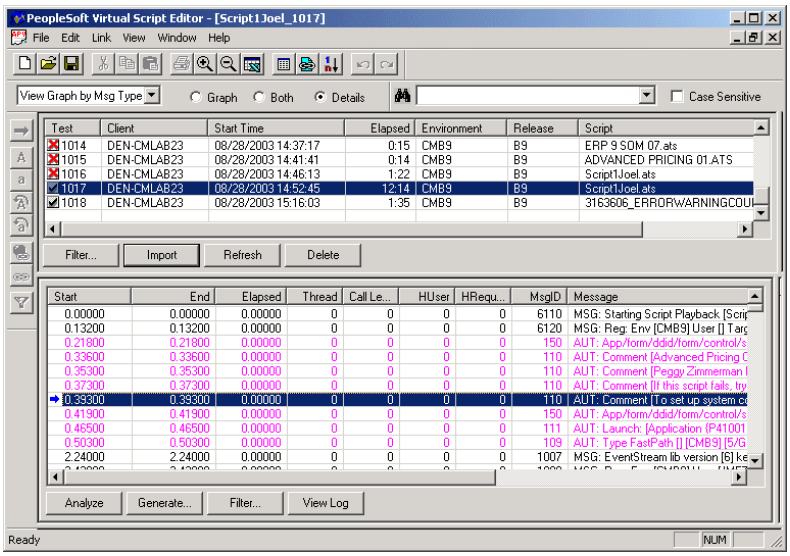
You click the Import button on the Virtual Script Editor form to import the results from an AutoPilot script playback session. The Virtual Script Editor populates the event pane with the event stream. The event stream contains a time stamping of each event. Therefore, you can review AutoPilot events 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:

Abbreviation in Message Column of Event Stream	**Type of Script Event**
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.

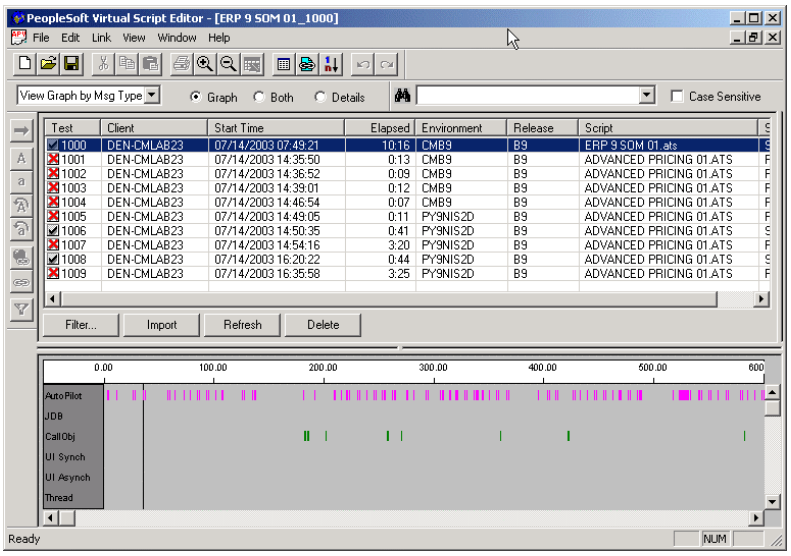
Filter Allows you to remove unwanted events from the list by applying criteria found in the Filter form.

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.

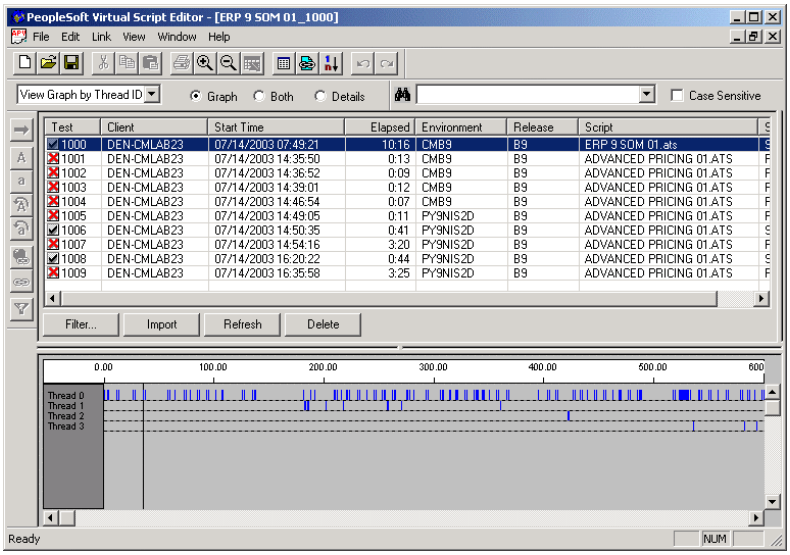
Event Graph

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, as shown below:



You also can break up the chronology by thread ID, as shown below:



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.

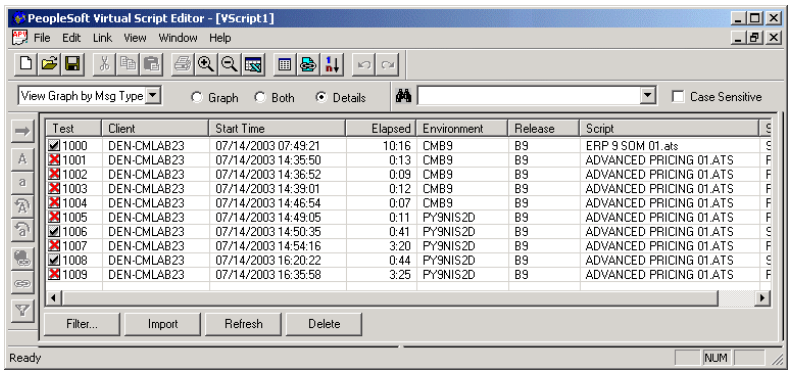
Finally, many API calls contain a request handle that points to a particular place in memory that the run-time engine has allocated for the call. The parameter for the request handle appears in the parameter detail pane if the API call used a request handle.

The ability of the AutoPilot and J.D. Edwards 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:

Column	Description
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	J.D. Edwards 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:

Virtual Script Editor Form Button	Purpose
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 J.D. Edwards 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.

In essence, value linking simulates the application logic that is used to run J.D. Edwards 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.

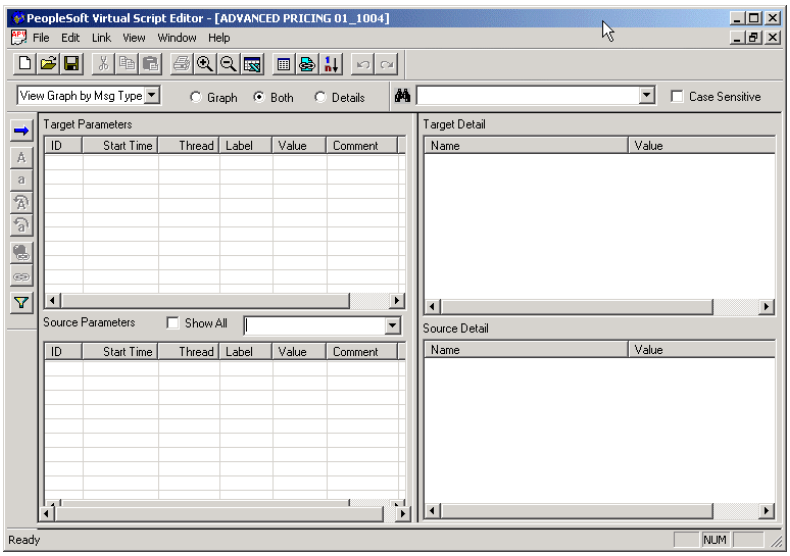
Information about the source and target parameters appears in separate panes of the Virtual Script Editor form. Each of the panes contains the following information about API calls made during script playback:

Column Heading in Target Parameters Pane and in Source Parameters Pane	Information Displayed
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 J.D. Edwards 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

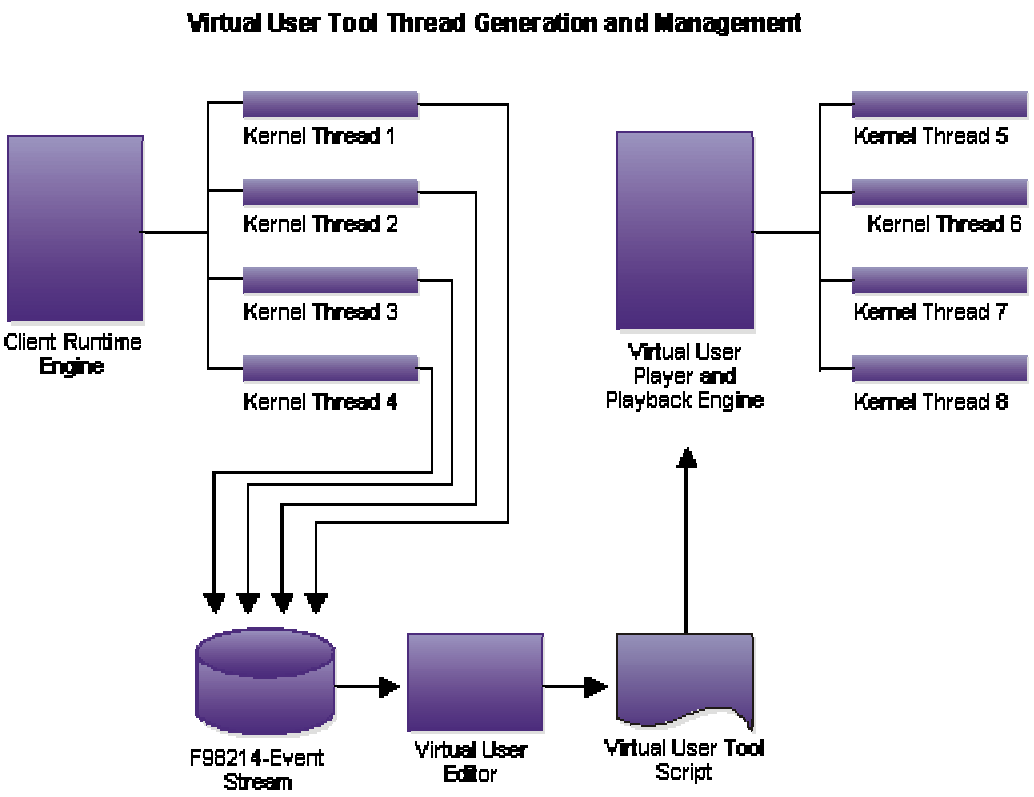
The Virtual Script Editor also stores the idThread numbers that AutoPilot gathers into the event stream during script playback. These identifier numbers represent the synchronous and asynchronous threads generated by the run-time engine. The run-time engine assigns each event to a thread and tags each 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 OneWorld Virtual User Tool's thread management strategy.

Note

During Virtual User Tool script playback, the Virtual Script Player renumbers 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

- *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:

Functions	Includes JDB and CallObject APIs
Assignment statements	Refers to values typed in AutoPilot
Conditional	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 J.D. Edwards 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 J.D. Edwards software users. It bypasses the presentation layer of J.D. Edwards software and reproduces the J.D. Edwards software 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 J.D. Edwards software run-time 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 OneWorld 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.
- OneWorld 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 OneWorld Virtual AutoPilot playback events and messages	LogDirectory	*c:\autopilot\VAP_LO*
OneWorld 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
1TooLate	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
1Timeout	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

- *Launching and Managing Multiple Script Playback* in the *Virtual AutoPilot Guide*
- *Playback Timing* in the *Virtual AutoPilot Guide*

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`	J.D. Edwards 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 J.D. Edwards 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

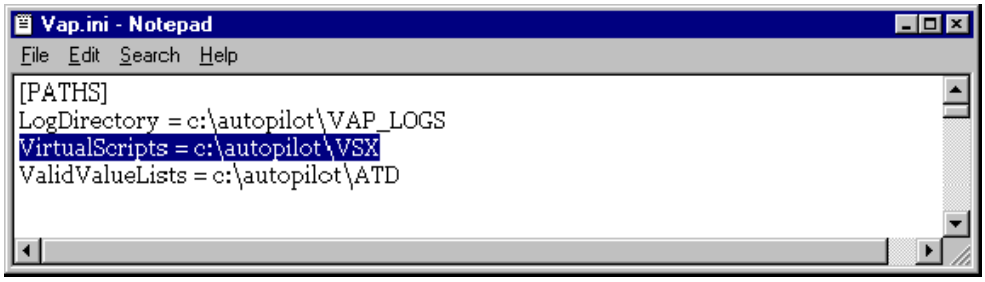
After you create a number of Virtual AutoPilot scripts, the VSMEditor allows you to concatenate any number of those scripts into a single master script. Concatenating single scripts into a single master script is advantageous because you can run a series of unrelated tasks during testing.

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

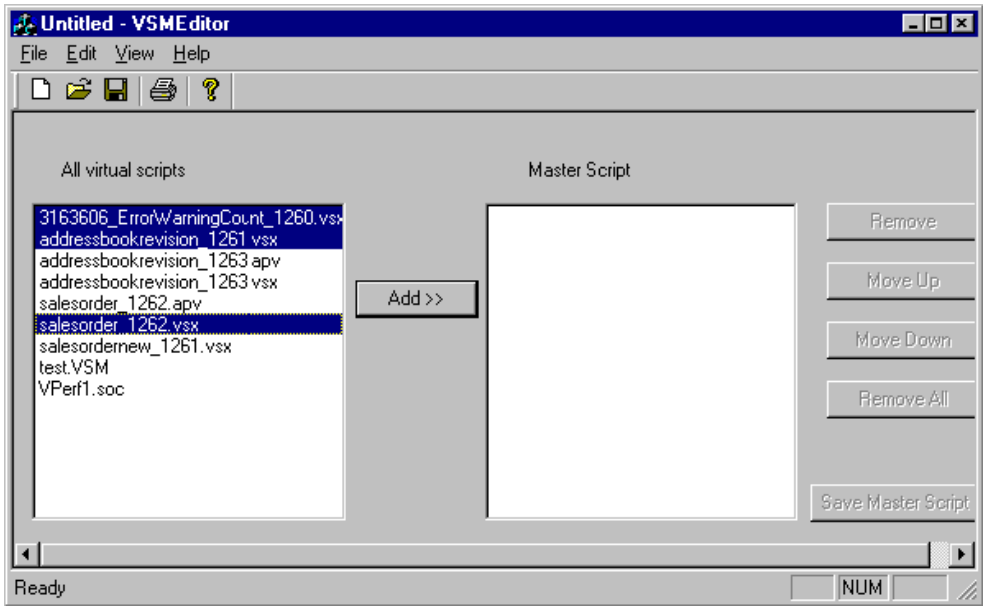
All Virtual Scripts List Box

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`.

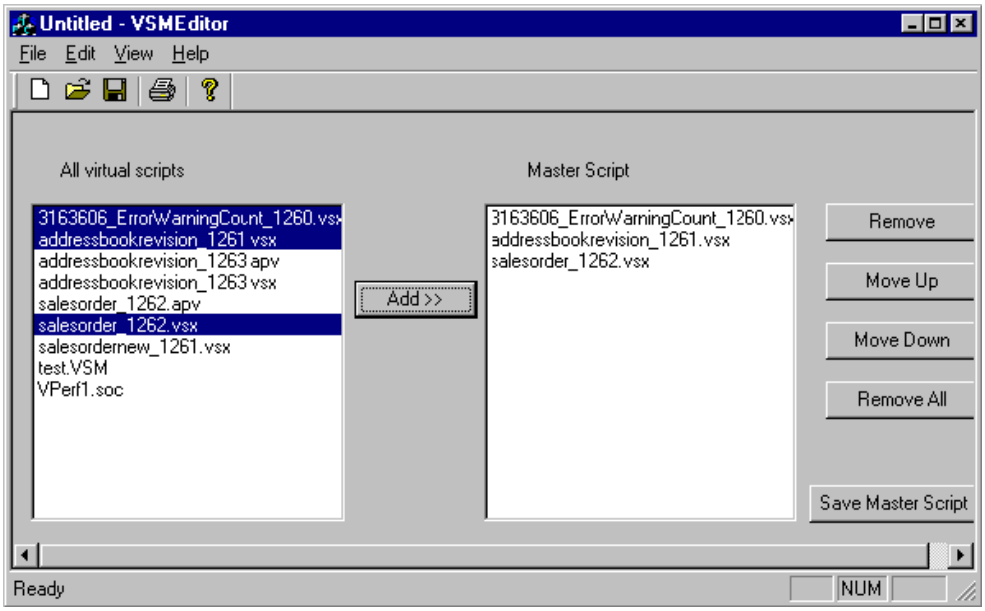
You enter the path to the location of your virtual scripts to set the VirtualScripts parameter:



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.



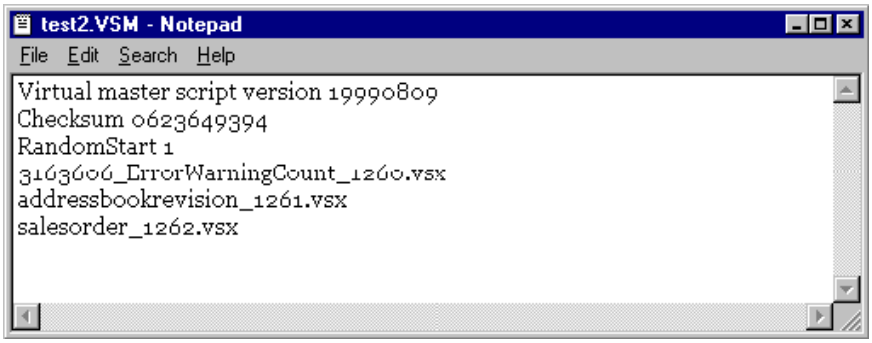
Master Scripts List Box

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 `.vsm` 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 are 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:

Virtual Runner Toolbar Button	Button Function
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 J.D. Edwards software as it handles the workload generated by many users. To achieve this goal, you use two key components of J.D. Edwards 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, and you can 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 J.D. Edwards 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 J.D. Edwards 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 on 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.

If this message appears, you should recapture the data, making sure that you sign on to the system through AutoPilot. To do so, close J.D. Edwards 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 J.D. Edwards 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.

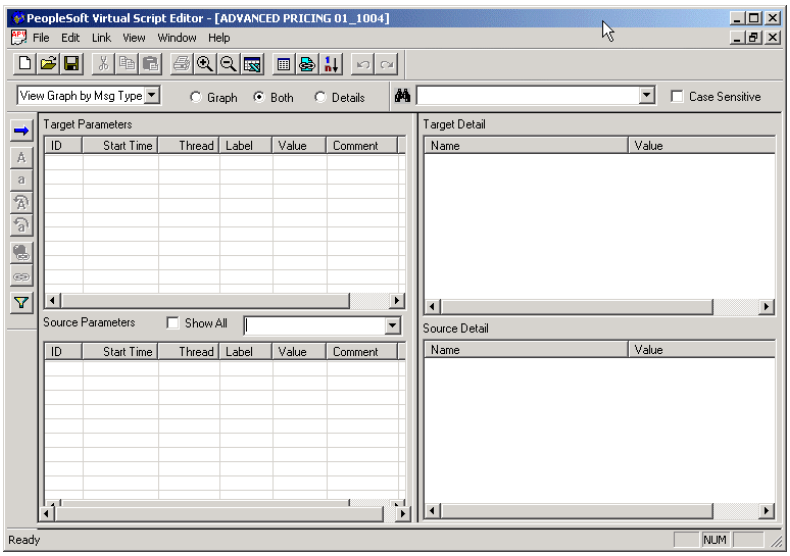
Note

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 J.D. Edwards 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

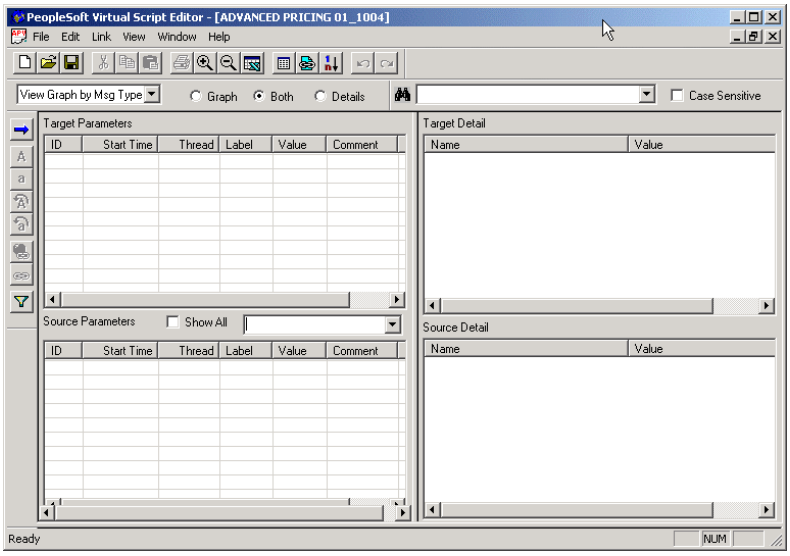
From your desktop or the appropriate directory, double-click the Virtual Script Editor executable.

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.



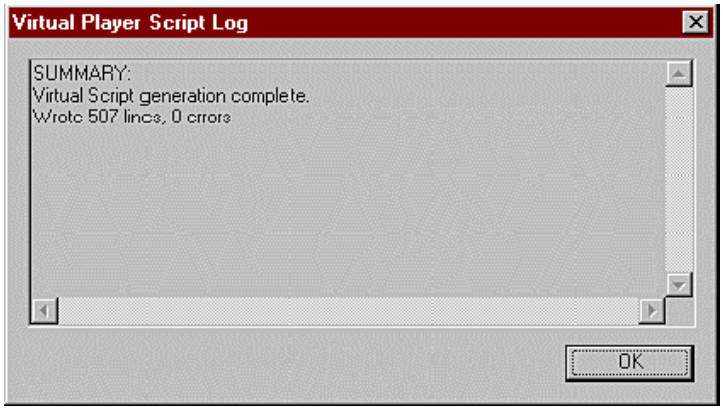
After Virtual AutoPilot generates the script, the Virtual Player Script Log form appears and displays information about the script generation, including the status (complete or incomplete), the number of lines, and the number of errors. If the Script Log form indicates that errors occurred during generation, 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 VSMEditor 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 VSMEditor 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.

   VSMEditor 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 VSMEditor 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.

Before You Begin

- 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 J.D. Edwards 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:

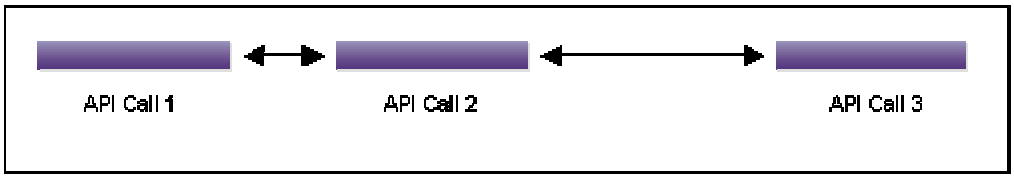
- Manage changes in the duration of individual API calls and the lengths of time between these calls within a single thread. This involves accurately simulating the process time required as the run-time engine handles user load.
- Handle timing differences that might affect interthread dependencies. 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

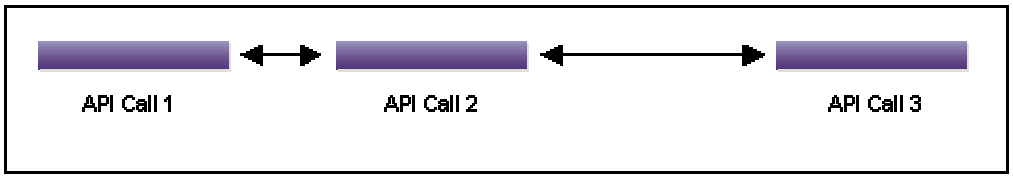


Preserving these chronological gaps during data capture provides the basis for simulating playback by many users, a situation that is likely to increase the length of time that is required to execute the same API calls.

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

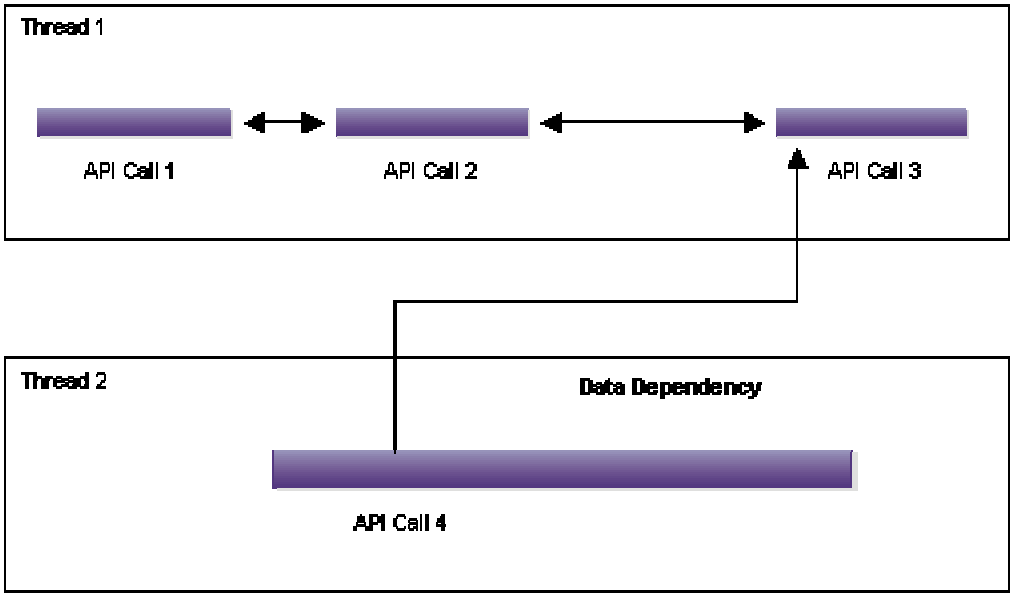


AutoPilot's ability to record the duration and length of time between API calls is also important because it can accurately determine the number of virtual users who can be simulated on a single workstation. For example, lengthy API calls might indicate an underpowered server, a workstation lacking the 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

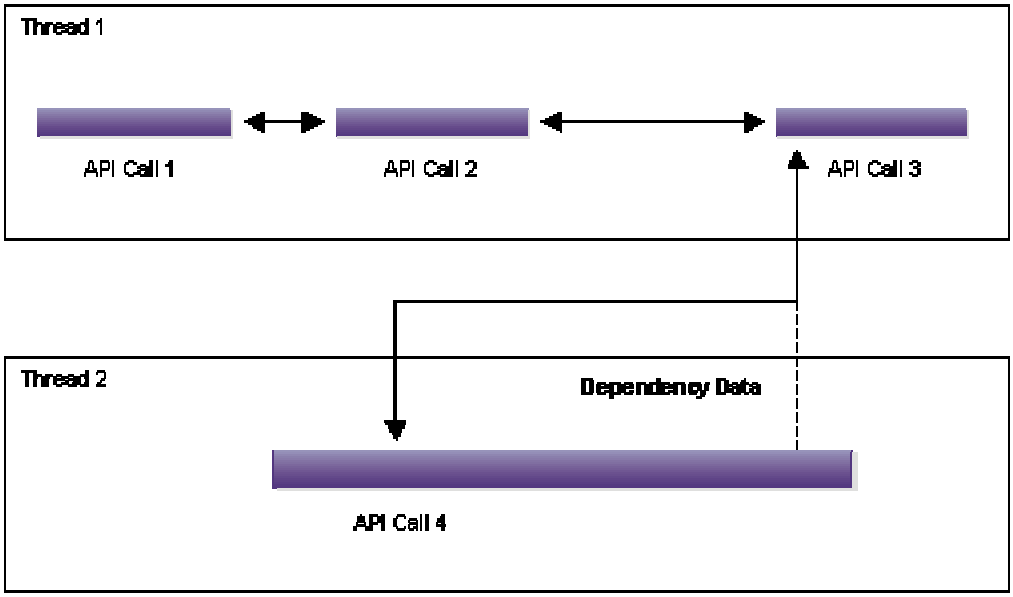
The Virtual Script Editor also plays a role in handling script playback timing so that Virtual AutoPilot can simulate a stressed environment. The run-time engine might, for example, create a thread that contains an API call with a data dependency on another call, which might, in turn, exist in a separate, 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



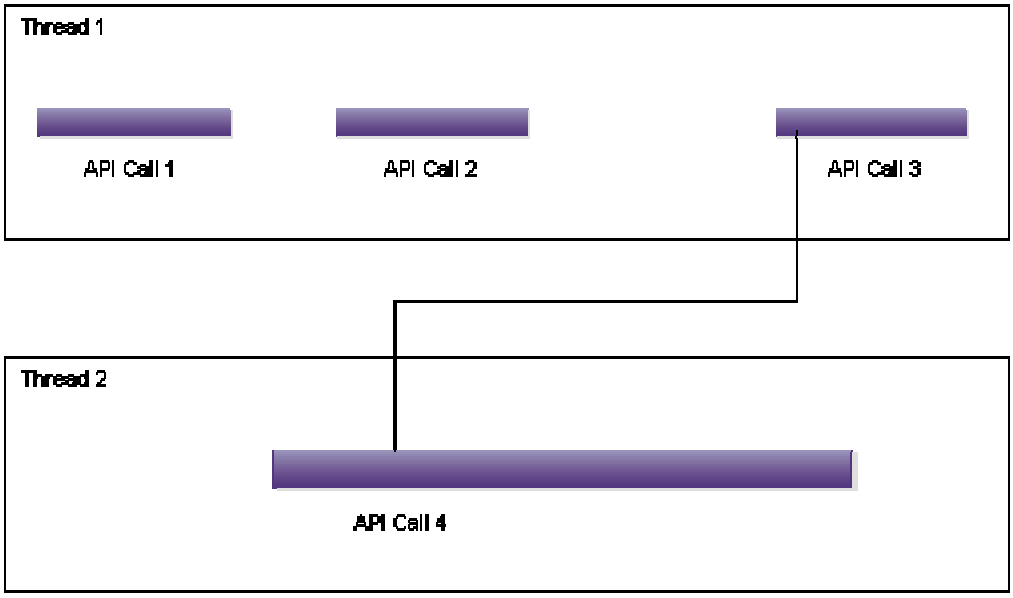
In a stressed environment, however, the duration of API calls might lengthen unpredictably. This might result in a data-dependent API call in one thread starting before the API upon which it depends has finished.

Stressed Playback in Virtual User Tool



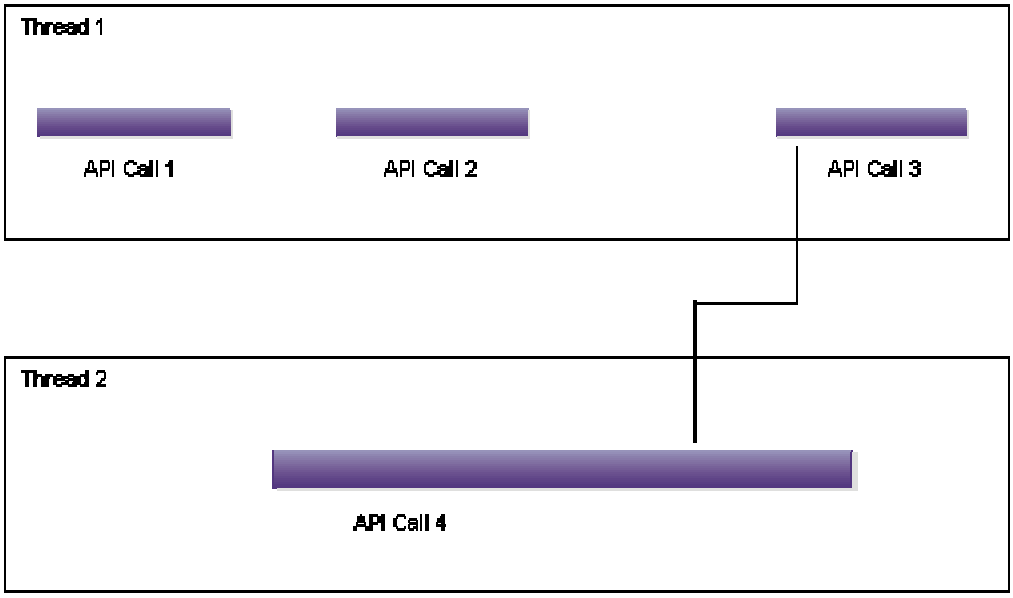
To deal with this potential problem, the Virtual Script Editor notes the data dependency when you forge value links between two API calls and preserves the timing interval between the calls.

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 J.D. Edwards software operations, Virtual AutoPilot must be able to manage synchronous and asynchronous API calls, an important management responsibility of the run-time 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 J.D. Edwards 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 run-time 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 run-time 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 J.D. Edwards 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 run-time engine uses, thereby managing interthread synchronization as well as event timing within threads.

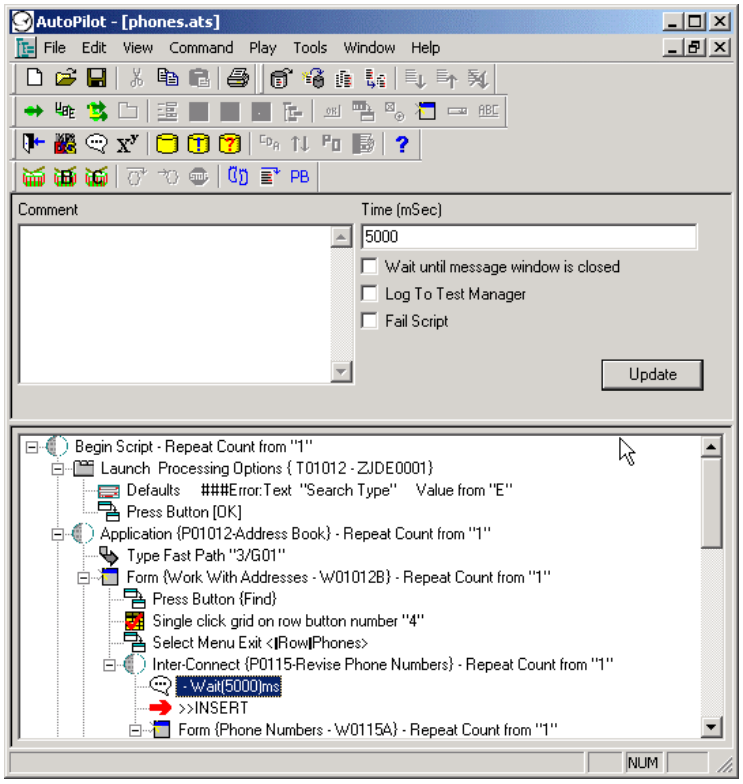
Synchronous and asynchronous call management provides another example of the Virtual User Tool's ability to accurately simulate the system, thereby providing you with a realistic picture of network and server stress.

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 J.D. Edwards software. You click the Wait Before Proceeding button and insert pauses in the playback in millisecond increments.



One possible reason for inserting wait periods in the script is that you might want 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 J.D. Edwards 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 in 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 that was 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.

- Locate the source of the script failure, which might be in either Virtual AutoPilot or the system.
- 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.

- 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*

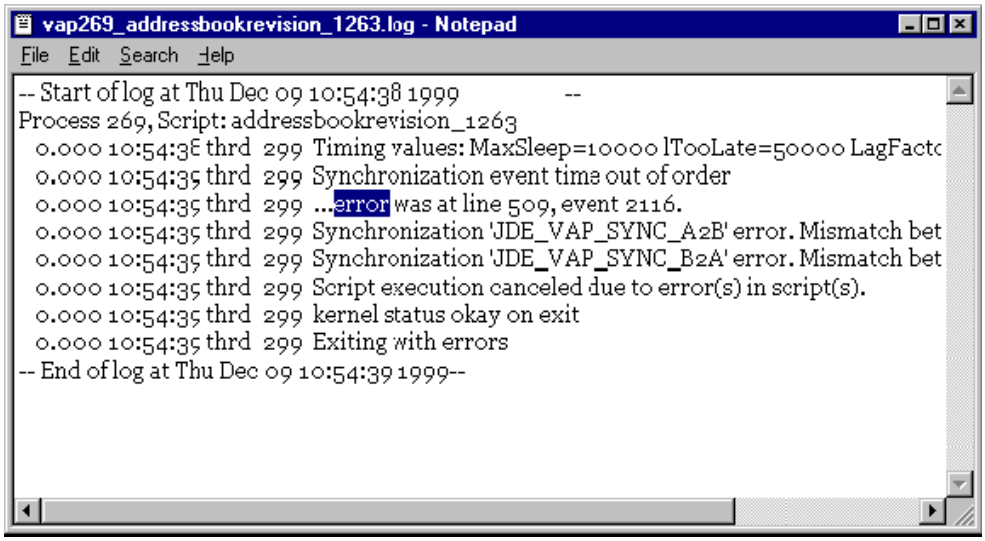
Finding Error Entries in the Virtual AutoPilot Log File

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:



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.

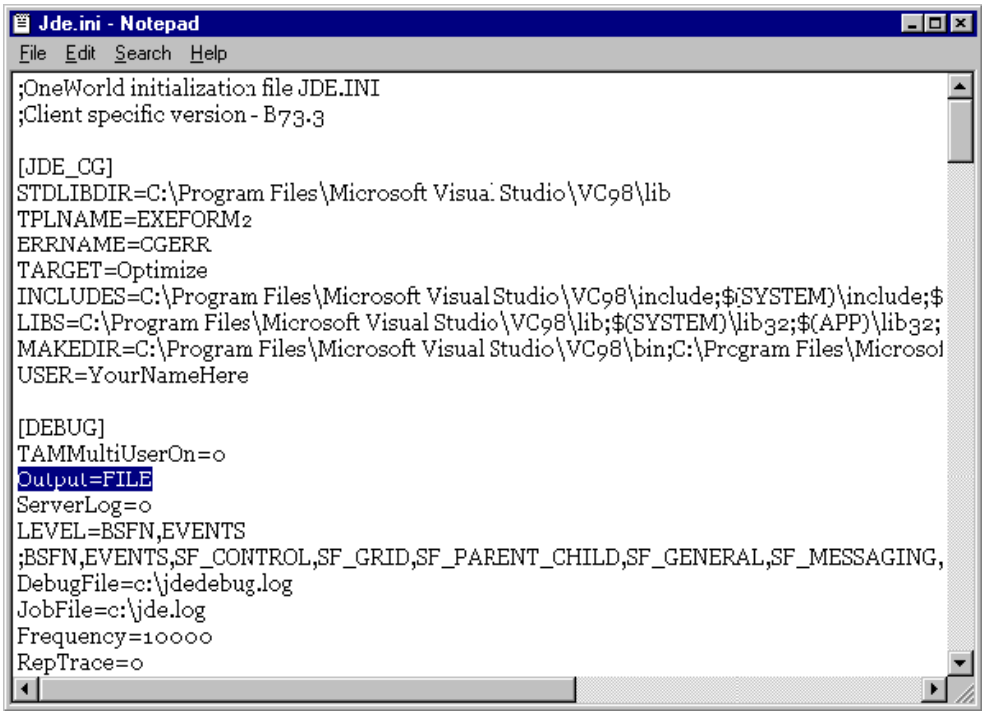
Investigating System Errors

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.



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 J.D. Edwards, you likely have a system problem that you must debug. Your debugging efforts might include a call to J.D. Edwards 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 J.D. Edwards form when an error occurs. When you perform this action, the system displays explanatory text, including possible causes and solutions, which 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 OneWorld 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 OneWorld 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 OneWorld

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. In OneWorld, right-click inside a OneWorld 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.

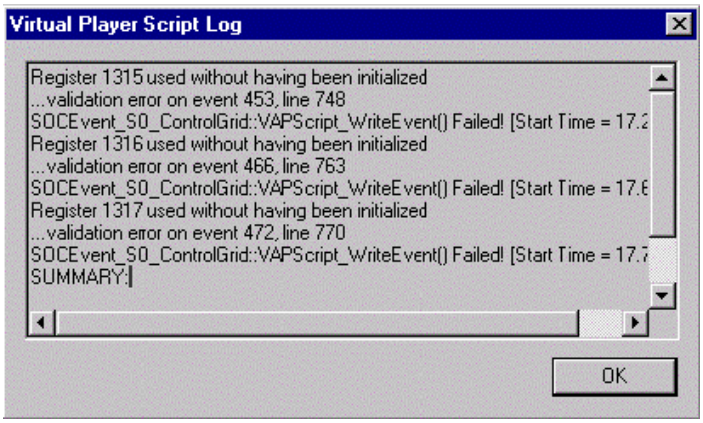
Identifying and Correcting Variable Value-Linking Errors

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 `jdedebug.log` file.

   If you have duplicate key errors, you will find them in the `jdedebug.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.

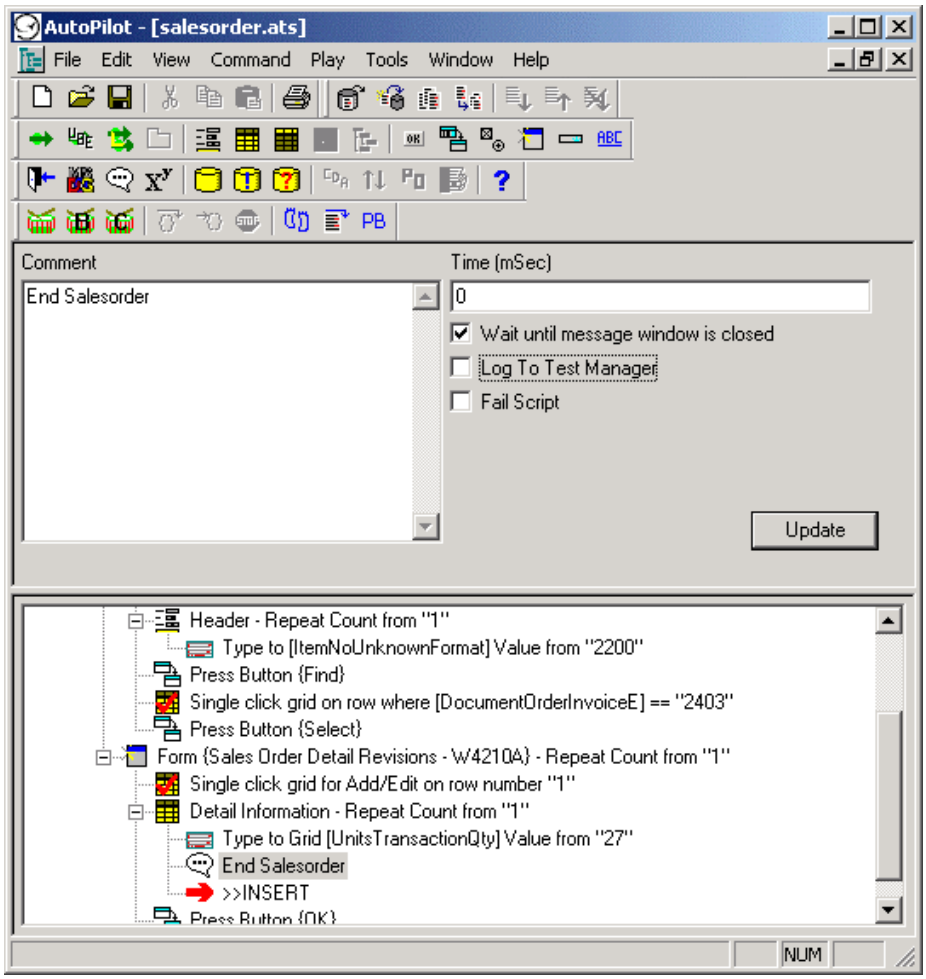
► To rectify irregular transaction times

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]` section `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 `jdedebug.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
`jde.log` and `jdedeb.log` messages fill up disk quickly during Virtual AutoPilot script playback	In `[DEBUG]` section of the `JDE.INI` file, set Output parameter to NONE
Virtual AutoPilot log file fills with messages, consuming disk space	In `[LOG]` section of the `vap.ini` file, set `MessageLevel` 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 J.D. Edwards 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.