



stonebranch
you imagine IT. we automate IT.

Opwise Universal Agent 5.2.0

Reference Guide

© 2015 by Stonebranch, Inc. All Rights Reserved.

1. Opwise Universal Agent 5.2.0 Reference Guide	3
1.1 Opwise Universal Agent - Fault Tolerance Implementation	4
1.1.1 Network Fault Tolerance - Universal Command	5
1.1.2 Network Fault Tolerance - Universal Connector	6
1.1.3 Network Fault Tolerance - Universal Data Mover	7
1.1.4 Manager Fault Tolerance - Universal Command	9
1.1.4.1 Manager Fault Tolerance - Universal Command - Functionality	10
1.1.4.2 Manager Fault Tolerance - Universal Command - Component Management	17
1.1.5 Client Fault Tolerance - Universal Connector	19
1.1.5.1 Client Fault Tolerance - Universal Connector - Modes	20
1.1.5.2 Client Fault Tolerance - Universal Connector - Parameters	21
1.1.5.3 Client Fault Tolerance - Universal Connector - Command ID Job Step	22
1.1.5.4 Client Fault Tolerance - Universal Connector - Command Identifier	23
1.1.5.5 Client Fault Tolerance - Universal Connector - Requesting Restart	24
1.1.5.6 Sample Command Lines For Working With Client Fault Tolerance	25
1.1.5.6.1 Working With Job Definition Files	26
1.1.5.6.2 Working With Pre-defined SAP Jobs	30
1.1.6 Implementing Fault Tolerance - Examples	34
1.1.6.1 Implementing Manager Fault Tolerance for Windows	35
1.2 Opwise Universal Agent - Network Data Transmission	36
1.2.1 SSL (Secure Socket Layer) Protocol	37
1.2.2 Universal V2 Protocol	39
1.2.3 Universal Application Protocol	40
1.2.4 Network Data Transmission Tuning	42
1.2.5 Network Data Transmission Configurable Options	44

Opswise Universal Agent 5.2.0 Reference Guide

The Opswise Universal Agent 5.2.0 Reference Guide contains *technical* information that is both:

- Specific to Opswise Universal Agent **or** Opswise Managed File Transfer
- Common to Opswise Universal Agent **and** Opswise Managed File Transfer

For *user* information specific to Opswise Universal Agent and Opswise Managed File Transfer, see:

- [Opswise Universal Agent 5.2.0 User Guide](#)
- [Opswise Managed File Transfer 5.2.0 User Guide](#)

Opwise Universal Agent - Fault Tolerance Implementation

Fault Tolerance Implementation

For Opwise Universal Agent, fault tolerance is the capability of its Opwise Universal Agent components to recover or restart from an array of error conditions that can occur in any large IT organization.

Errors occur as a result of human, software, or hardware conditions. The more resilient a product is to errors, the greater value it offers.

Currently, fault tolerance is implemented in three Opwise Universal Agent components:

- Universal Command
- Universal Connector
- Universal Data Mover

Detailed Information

The following pages provide detailed information for Fault Tolerance Implementation:

- [Network Fault Tolerance - Universal Command](#)
- [Network Fault Tolerance - Universal Connector](#)
- [Network Fault Tolerance - Universal Data Mover](#)
- [Manager Fault Tolerance - Universal Command](#)
- [Client Fault Tolerance - Universal Connector](#)
- [Implementing Fault Tolerance - Examples](#)

Network Fault Tolerance - Universal Command

- [Overview](#)
- [Network Fault Tolerant Protocol](#)
- [Universal Command Manager](#)
- [Universal Command Server](#)

Overview

Universal Command uses the TCP/IP protocol for communications over a data network. The TCP/IP protocol is a mature, robust protocol capable of resending packets and rerouting packets when network errors occur. However, data networks do have problems significant enough to prevent the TCP/IP protocol from recovering. As a result, the TCP/IP protocol terminates the connection between the application programs.

As with any application using TCP/IP, Universal Command is subject to these network errors. Should they occur, a product can no longer communicate and must shut down or restart. These types of errors normally show themselves as premature closes, connection resets, time-outs, or broken pipe errors.

Network Fault Tolerant Protocol

Universal Command provides the ability to circumvent network errors with its Network Fault Tolerant protocol. By using this protocol, Universal Command traps the connection termination caused by the network error and reestablishes the network connections. When connections have been reestablished, processing resumes automatically from the location of the last successful message exchange. No program restarts are required and no data is lost.

The Network Fault Tolerant protocol acknowledges successfully received messages and checkpoints successfully sent messages. This reduces data throughput. Consequentially, the use of network fault tolerance should be weighed carefully in terms of increased execution time versus the probability of network errors and cost of such errors. For example, it may be easier to restart a program than to incur increased execution time.

When a network connection terminates, the Universal Command Manager enters a network reconnect phase. In this phase, the Manager attempts to connect to the Universal Command Server and reestablish its network connections. The condition that caused the network error can persist only for seconds, or it can persist for days.

The Manager attempts Server reconnection for a limited amount of time, as specified by the following configuration options:

- [RECONNECT_RETRY_COUNT](#) (number of retry attempts)
- [RECONNECT_RETRY_INTERVAL](#) (frequency of retry attempts)

If all attempts fail, the Manager ends with an error.

When a network connection terminates, the Server's action depends on whether or not it is executing with [Manager Fault Tolerance](#).

Without Manager Fault Tolerance, the Server enters a disconnected state and waits for the Manager to reconnect. The user process continues running. However, if the user process attempts any I/O on the standard files, it will block. The Server waits for the Manager to reconnect for a period of time defined by the Manager's [RECONNECT_RETRY_COUNT](#) and [RECONNECT_RETRY_INTERVAL](#) configuration options. When that time has expired, the Server terminates the user process and exits.

With Manager Fault Tolerance, the Server continues executing in a disconnected state. The Server satisfies all user process standard I/O requests. The user process does not block. It continues to execute normally. When the user process ends, the Server waits for a Manager reconnect for a period of time defined by the [JOB_RETENTION](#) configuration option.

Universal Command Manager

You can configure Universal Command Manager to request the use of the Network Fault Tolerant protocol via its [NETWORK_FAULT_TOLERANT](#) configuration option.

If the Server does not support the protocol or is not configured to accept the protocol, the Manager continues without using the protocol.

Universal Command Server

You can configure Universal Command Server with or without the Network Fault Tolerant protocol via its [NETWORK_FAULT_TOLERANT](#) configuration option.

If the Server is configured with the protocol off, the Manager cannot override it. If the Server is configured with the protocol on, the Manager [NETWORK_FAULT_TOLERANT](#) configuration option specifies whether or not the protocol is actually used.

Network Fault Tolerance - Universal Connector

- [Overview](#)
- [Points of Failure](#)
- [Network Fault Tolerance Configuration Parameters](#)

Overview

Universal Connector commands are processed by calling appropriate BAPI functions in the SAP system. The BAPI function calls are issued over an RFC connection. Universal Connector provides fault tolerance at the RFC level. If an RFC call fails, the call is retried until it completes successfully or exceeds a user-definable retry limit.

If an RFC call fails, Universal Connector will close the current RFC connection and establish a new RFC connection in order to continue processing. The process of establishing and preparing an RFC connection is referred to here as the RFC logon process. The RFC logon process involves establishing an RFC connection, logging on to the XMI interface and setting the XMI audit level. If the RFC logon process fails, it will be retried until it completes successfully, or exceeds a user definable retry limit. When the new RFC connection is established successfully, Universal Connector will reissue the failed RFC call.

The entire process of establishing a new RFC session and reissuing the failed RFC call will be retried until either:

- RFC call completes successfully.
- User-definable RFC retry limit is exceeded.

Some BAPI functions should not be retried in an unknown state; they are points of failure within the Universal Connector fault tolerant solution.

Points of Failure, below, lists the points of failure and their relationship to Universal Connector commands.

Points of Failure

The points of failure within Universal Connector fault tolerant architecture are:

- Job Submission
- Job Modification
- Job Start

Some BAPI functions called in these processes cannot be restarted in an unknown state without possible negative consequences. If an RFC call fails issuing those BAPIs, Universal Connector will end unsuccessfully.

Network Fault Tolerance Configuration Parameters

The following set of Universal Connector configuration options can be used to fine-tune the fault tolerance support for a particular environment:

- [LISTEN_INTERVAL](#) (-rfc_listen_interval)
- [LOGON_RETRY_COUNT](#) (-rfc_logon_retry_count)
- [LOGON_RETRY_INTERVAL](#) (-rfc_logon_retry_interval)
- [SECURE_CFT](#) (-rfc_retry_count)
- [RETRY_CALL_INTERVAL](#) (-rfc_retry_interval)
- [TIMEOUT_INTERVAL](#) (-rfc_timeout)

See [RFC \(Remote Function Call\) Options](#) for details concerning the use of these parameters.

Network Fault Tolerance - Universal Data Mover

- [Overview](#)
- [Network Fault Tolerance](#)
 - [Open Retry](#)
 - [Component Management](#)
 - [Communication State Values](#)

Overview

For Opwise Managed File Transfer, fault tolerance is the capability of its components to recover or restart from an array of error conditions that occur in any large IT organization.

Errors occur as a result of human, software, or hardware conditions. The more resilient a product is to errors, the greater value it offers.

Currently, network fault tolerance is implemented in one Opwise Managed File Transfer component:

- [Universal Data Mover](#)

Network Fault Tolerance

UDM uses the TCP/IP protocol for communications over a data network. The TCP/IP protocol is a mature, robust protocol capable of resending and rerouting packets when network errors occur. However, data networks do have problems significant enough to prevent the TCP/IP protocol from recovering. As a result, the TCP/IP protocol terminates the connection between the application programs. Like any application using TCP/IP, UDM is subject to these network errors. Should they occur, a product can no longer communicate and must shutdown or restart. These types of errors normally show themselves as premature closes, connection resets, time-outs, or broken pipe errors.

UDM provides the ability to circumvent these types of errors with its Network Fault Tolerant protocol. By using the network fault tolerant protocol, UDM traps the connection termination caused by the network error and it reestablishes the network connections. Once connections are reestablished, processing automatically resumes from the location of the last successful message exchange. No program restarts are required and no data are lost.

The Network Fault Tolerant protocol acknowledges and checkpoints successfully received and sent messages, respectively. The network fault tolerant protocol does reduce data throughput. Consequentially, the use of network fault tolerance should be carefully weighed in terms of increased execution time versus the probability of network errors and cost of such errors. For example, it may be easier to restart a program than to incur increased execution time.

When a network connection terminates, the UDM Manager will enter a network reconnect phase. In the reconnect phase, the Manager attempts to connect to the UDM Server and reestablish its network connections. The condition that caused the network error can persist for seconds or days. The Manager will attempt Server reconnection for a limited amount of time (configured with the [RECONNECT_RETRY_COUNT](#) and [RECONNECT_RETRY_INTERVAL](#) configuration options). These two options specify, respectively, how many reconnect attempts are made and how often they are made. After all attempts have failed, the Manager ends with an error.

When a network connection terminates, the Server enters a disconnected state and waits for the Manager to reconnect. The user process continues running; however, if the user process attempts any I/O on the standard files, it will block. The Server waits for the Manager to reconnect for a period of time defined by the Manager's [RECONNECT_RETRY_COUNT](#) and [RECONNECT_RETRY_INTERVAL](#) configuration options. Once that time has expired, the Server terminates the user process and exits.

UDM can request the use of the Network Fault Tolerant protocol. If the Server does not support the protocol or is not configured to accept the protocol, the Manager continues without using the protocol.

The [NETWORK_FAULT_TOLERANT](#) and [RECONNECT_RETRY_INTERVAL](#) option is used to request the protocol.

Open Retry

Open Retry is a type of fault tolerance used at the session-establishment level.

(Network fault tolerance is used from the time that a session has been fully established until the session has terminated.)

Open Retry is used during the establishment phase of a session. UDM tries to establish a session when the `open` command is issued. If the [OPEN_RETRY](#) configuration option value is **yes**, and UDM fails to establish the session due to a network error, timeout, or the inability to start a transfer server, it will retry the `open` command based on the settings of the [OPEN_RETRY_COUNT](#) and [OPEN_RETRY_INTERVAL](#) configuration options.

Component Management

In order to fully understand UDM fault tolerant features, some understanding of how the Universal Broker manages components is necessary.

Universal Broker manages component start-up, execution, and termination. Universal Broker and its components have the ability to communicate service requests and status information between each other.


Universal Broker maintains a database of components that are active or have completed and waiting for restart or reconnection. The component information maintained by Universal Broker determines the current state of the component. This state information is required by Universal Broker to determine whether or not a restart or reconnect request from a Manager is acceptable. The ;Universal Broker component information can be viewed with the [Universal Query](#) utility.

One bit of component information maintained by Universal Broker is the component's communication state. The communication state primarily determines what state the Universal Data Mover Server is in regarding its network connection with a Manager and the completion of the user process and its associated spooled data.

Communication State Values

The following table describes the communication state values.

- **Reconnect** column indicates whether or not a network reconnect request is valid.
- **Restart** column indicates whether or not a restart request is valid.

State	Reconnect	Restart	Description
COMPLETED	NO	NO	Server and manager have completed. All standard output and standard error files have been sent to the manager and the user process's exit status.
DISCONNECTED	YES	YES	<p>Server is not connected to the Manager. This occurs when a network error has occurred, the Manager halted, or the Manager host halted.</p> <p>The Server is either executing with the Network Fault Tolerant protocol, is restartable, or both.</p> <div style="background-color: #ffffcc; padding: 5px; border: 1px solid #ccc;"> <p> Note The Server cannot tell if the Manager is still executing or not, since it cannot communicate with the Manager.</p> </div>
ESTABLISHED	NO	NO	Server and Manager are connected and processing normally. This is the most common state when all is well.
RECONNECTING	NO	NO	<p>Server has received a reconnect request from the Manager to recover a lost network connection.</p> <p>This state should not remain long, only for the time it takes to re-establish the network connections.</p>
STARTED	NO	NO	<p>Server has started.</p> <p>If the Server is restartable it is receiving the standard input file from the Manager and spooling it.</p>

Manager Fault Tolerance - Universal Command

Overview

Distributed applications are comprised of many independent components running on host systems, throughout the enterprise, connected with a data network. Many of the host systems are in different physical locations, in different organization groups, and have different system management policies. It can be difficult to schedule individual host downtime when there are so many overlapping requirements.

Host systems must be shut down at scheduled intervals and, unfortunately, at unscheduled intervals. The impact of a system being down must be minimized by a distributed application.

With the Manager Fault Tolerant feature, Universal Command components (Managers and Servers) can be shut down and restarted at a later time. After a Manager has been started, it can be terminated and restarted at a later time. It can be shut down for any period of time: seconds, days, or even months.

Detailed Information

The following pages provide detailed information for Manager Fault Tolerance - Universal Command:

- [Functionality](#)
- [Component Management](#)

Manager Fault Tolerance - Universal Command - Functionality

- Manager Fault Tolerance Functionality
- Command Identifier
- Standard I/O Files
- Requesting Restart
- Case Example 1 - Normal Execution
 - Components
 - Sequence of Events
- Case Example 2 - Restart when User Process is Executing
 - Sequence of Events
- Case Example 3 - Restart when User Process has Ended
 - Sequence of Events

Manager Fault Tolerance Functionality

The basic functionality of Manager fault tolerance is:

1. Manager requests the execution of a command on a remote system.
2. Command executes on the remote system, optionally reading and writing data.
3. Manager redirects:
 - Its standard input data to the standard input of the remote command.
 - Standard output file and standard error file of the remote command to its own standard output and standard error.

If the Manager is terminated or the Manager's host system is shut down, the remote command cannot read the Manager's standard input or write its standard output and error files. Without Manager fault tolerance, the remote command must terminate, since its data source and destination are now gone. Otherwise, it would wait forever.

Manager fault tolerance provides an execution environment in which the Manager is not required in order for the user process to continue execution on the remote system. The user process can execute to completion with or without a Manager connected.

When the Manager starts a user process, the Manager executes as normal; standard output and standard error files are redirected back to the Manager as the user process produces the data. The difference is data spooling. In order for the user process to have real-time access to its input and output, the data is spooled in the Universal Spool Database. The spool provides complete independence from the Manager. The spool subsystem satisfies all data requirements for the user process via the Universal Command Server.

The Manager can terminate and a new Manager can restart and reconnect to the user process. If the user process has completed, the new Manager receives the user processes standard files and its exit status. The restarted Manager behaves in all ways as if it was the originating Manager .

Command Identifier

A Manager requests Manager fault tolerance with the `MANAGER_FAULT_TOLERANT` configuration option and by providing a command identifier (command ID) using the `COMMAND_ID` configuration option. The command ID identifies the unit of work being executed. In this context, a unit of work includes the Manager, Server, and user process.

The Manager indicates to the Server that this request is restartable. The `COMMAND_ID` configuration option provides a command identifier that uniquely identifies the Server and user process on the remote host. When a Manager is restarted, it must provide the same command ID identifying the Server and user process with which it wants to reconnect.

Providing a unique command ID is not trivial. Many Managers may be executing on many different hosts, and all executing work on the same Server host. It is possible for a Manager to start a restartable command from one host, terminate, and restart on a completely different host.

The command ID value can be any text value of unrestricted length. In practical terms, the character set and limits on command line length of the Manager host impose restrictions on the value.

Standard I/O Files

The Universal Spool system satisfies all user process data requests via the Universal Command Server. When the user process reads from its standard input file, the Server reads it from the spool and provides it to the user process. When the user process writes to standard output or error, the Server receives the data and writes it to the spool.

A Manager requesting restart capability (Manager fault tolerance) first transfers its entire standard input file to the Server, which it in turns writes to the spool. When all data has been received, the Server creates the user process. This provides complete Manager independence for the entire life of the user process.

As long as the Manager is connected, the standard output and standard error files are transferred to the Manager, as the user process produces the data, all in real-time. The data also is written to the spool. If the Manager terminates, the data is written to the spool only.

A restarted Manager is sent all of the standard output and standard error files, from the beginning, that currently is spooled. If the user process still is executing, the restarted Manager will receive all of the data currently spooled. When it has caught up with the data being produced, the Manager starts to receive the data from the user process as it is written.

Requesting Restart

When a restartable Manager is initiated, it is either an initial instance or a restarted instance of a command ID. The command ID identifies a unit of work represented by the Manager, Server, and user process. See [Command Identifier](#), above, for more information on the command ID.

The **RESTART** configuration option specifies whether or not the Manager instance is requesting a restart of a previous command ID. Possible **RESTART** values are **yes**, **no**, and **auto**.

The **auto** value specifies:

- If there is no existing command ID executing on the remote host, consider this Manager execution the first instance.
- If there is an existing command ID, and it is not connected to any Manager, consider this a restart of the command ID.

The **auto** value permits automatic restart by eliminating the need to modify the **RESTART** value for the initial instance and restarted instance.

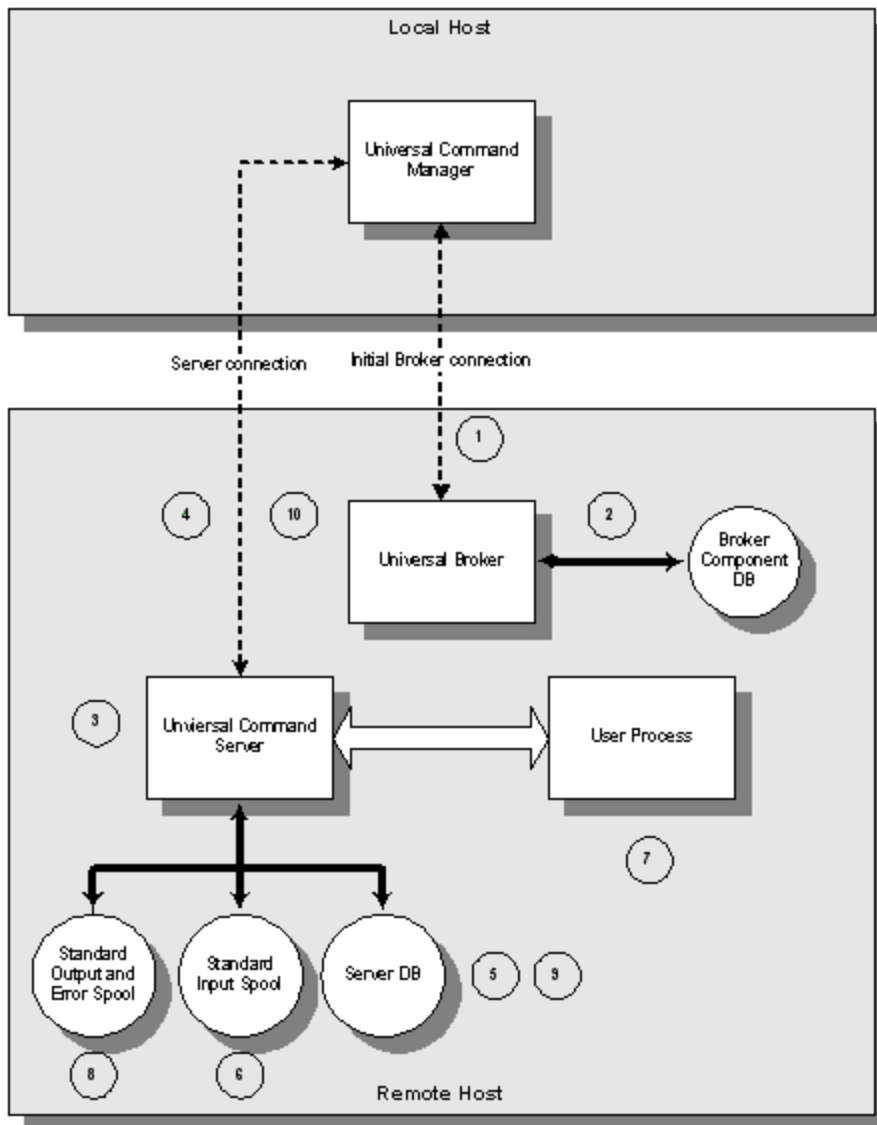


Note

The **auto** value cannot be used with a **COMMAND_ID** configuration option value of *, which specifies that the UCMD Manager will generate a unique command ID for each run.

Case Example 1 - Normal Execution

The following figure diagrams the sequence of events that occur when a restartable Manager requests the execution of a command on a remote host. In this case, the Manager and Server remain executing and connected until normal completion of the user process.



The Local Host is the host on which the Manager is being executed.

The Remote Host is the host on which the Manager is requesting command execution.

Components

The components involved are:

- **Universal Command Manager**
The Manager requests remote execution of a command or script. The Manager executes the remote command in a manner such that the command appears to be executed locally.
- **Universal Broker**
The Broker manages Opwise Universal Agent component execution.
- **Universal Command Server**
The Server executes the Manager requested command and processes the user process's standard I/O requests.
- **User Process**
The user process represents the Manager requested command.

Sequence of Events

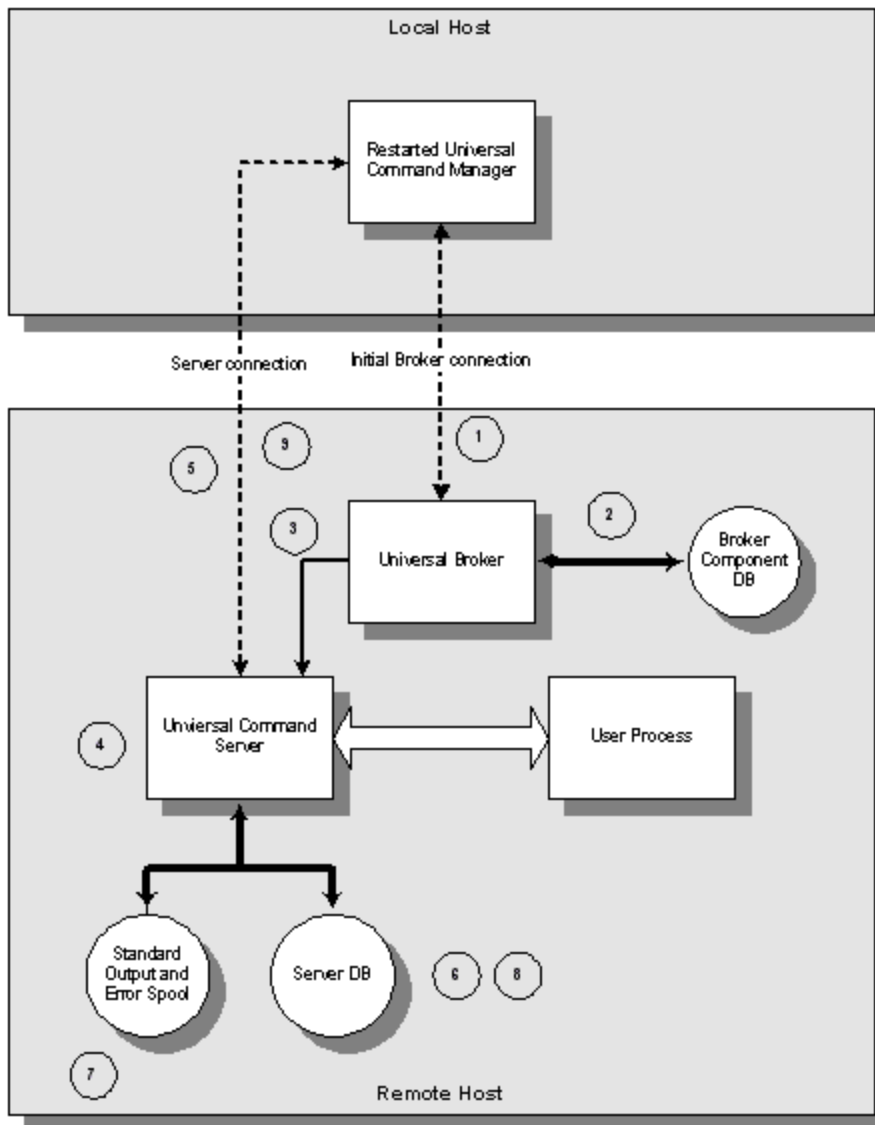
The diagram demonstrates the sequence of events that occur when a restartable Manager requests command execution on a remote host. The numbers enclosed in circles represent the sequence of events and correspond to the listed descriptions below.

Step 1	The Manager connects to the Broker and sends a request to start a Server. The start request from the Manager requests Manager fault tolerance and includes the command ID to identify the unit of work.
Step 2	The Broker records the unit of work in the Broker Component Database as restartable for possible future restarts.
Step 3	The Broker starts an instance of the Server.
Step 4	The Manager and Server exchange messages that specify all options used to carry out the request.
Step 5	The Server records the unit of work in the Universal Command Server Database for possible future restarts.
Step 6	The Manager sends all standard input data to the Server, and the Server writes the standard input data to the Universal Spool database.
Step 7	Once all standard input is spooled, the Server starts the user process.
Step 8	As the user process writes standard output and standard error data, the Server writes the data to the Universal Spool database. If the Manager is connected to the Server, the data is written to the Manager as well.
Step 9	The user process executes until completion. Once the user process completes, the Server writes the exit status of the user process to the Universal Command Server Database.
Step 10	The Server sends the exit status to the Manager. This completes the unit of work.

Case Example 2 - Restart when User Process is Executing

The following figure diagrams the sequence of events that occur when a Manager requests a restart of a currently executing unit of work. In this case the initial instance of the Manager terminated. A restarted instance of the Manager is started and requests to be reconnected to the unit of work.

This example continues from [Case Example 1 - Normal Execution](#); please refer to that example for details of the component descriptions included in the following diagram.



Sequence of Events

The diagram demonstrates the sequence of events that occur when a Manager requests to be restarted with a unit of work identified by a command ID. The numbers enclosed in circles represent the sequence of events and correspond to the listed descriptions below.

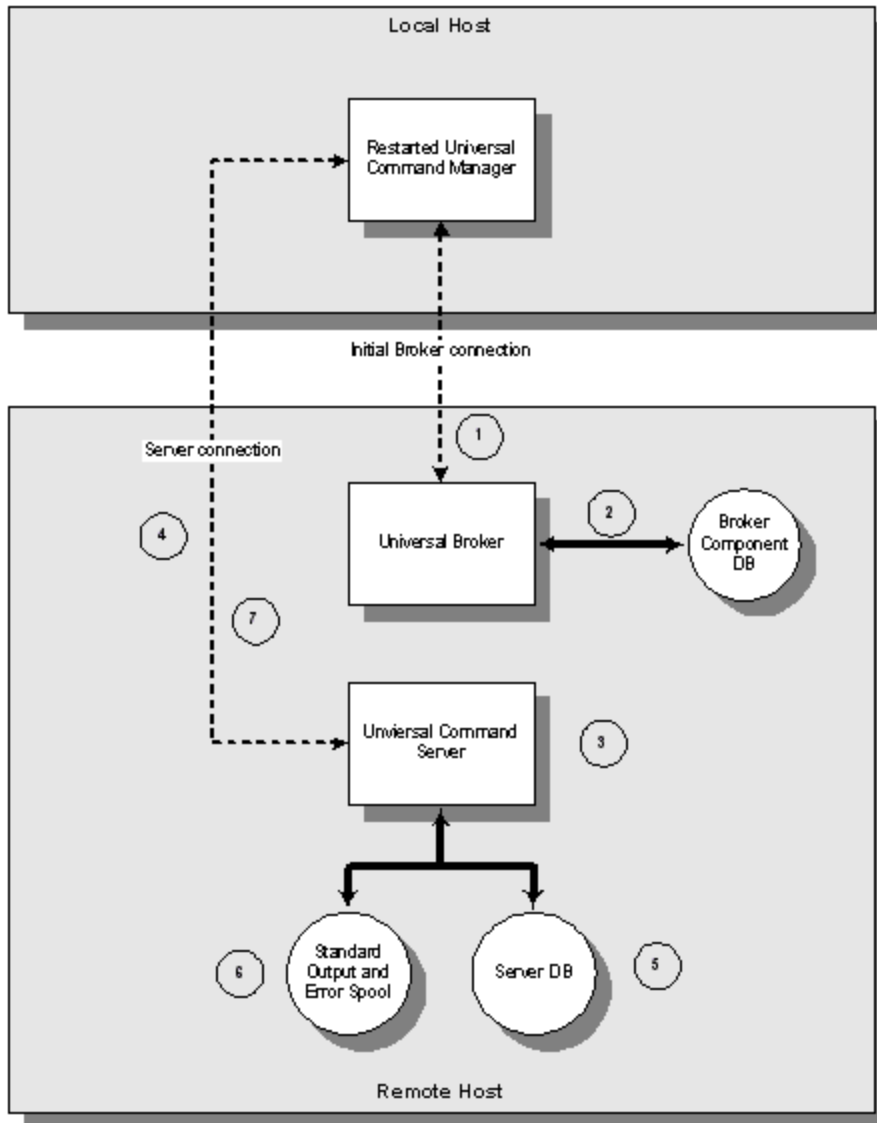
Step 1	The restarted instance of the Manager sends a restart request to the Broker. The restart request contains the command ID specified as part of the invocation of the Manager.
Step 2	The Broker verifies that the component is restartable and that the components communication state is acceptable for a restart request. If the Server component were currently connected to a Manager, its communication state would not permit a restart request.
Step 3	The Broker sends the restart request to the Server corresponding to the command ID.
Step 4	The Server authenticates the request with the Manager-supplied user ID and password. The password must be the same as the initial Manager instance.
Step 5	The Manager and Server exchange options that are used to carry out the request.
Step 6	The Server records the restart in the Universal Command Server Database.
Step 7	The Server sends spooled standard output and error files to the Manager. This is performed while the user process may still be writing standard output and error to the spool. Once all spooled output is sent to the Manager, the Server will send standard output and error from the user process as it is being produced.
Step 8	The user process executes to completion. The Server records the user process exit status in the Universal Command Server Database.

Step 9 The Server sends the exit status to the Manager. This completes the unit of work.

Case Example 3 - Restart when User Process has Ended

The following figure diagrams the sequence of events that occur when a Manager requests a restart of a unit of work that has completed. In this case, the initial instance of the Manager has terminated, the user process completed normally, and a restarted instance of the Manager is started and requests to be reconnected to the completed unit of work.

This example continues from [Case Example 1 - Normal Execution](#); please refer to that example for details of the component descriptions included in the following diagram.



Sequence of Events

The diagram demonstrates the sequence of events that occur when a Manager requests to be restarted with a unit of work identified by a command ID. The user process in this case has completed execution. The numbers enclosed in circles represent the sequence of events and correspond to the following descriptions:

Step 1	The restarted instance of the Manager sends a restart request to the Broker. The restart request contains the command ID specified as part of the invocation of the Manager.
Step 2	The Broker verifies that the component is restartable and that the components communication state is acceptable for a restart request. If the Server component were currently connected to a Manager, its communication state would not permit a restart request.

Step 3	Since the user process has completed, the Broker starts a new Server to process the restart request. The Server authenticates the request with the Manager-supplied user ID and password. The password must be the same as the initial Manager instance.
Step 4	The Manager and Server exchange options that are used to carry out the request.
Step 5	The Server records the restart in the Universal Command Server Database.
Step 6	The Server sends spooled standard output and error files to the Manager.
Step 7	The Server sends the user process exit status to the Manager. This completes the unit of work.

Manager Fault Tolerance - Universal Command - Component Management

Overview

In order to fully understand Universal Command fault tolerant features, some understanding of how the Universal Broker manages components is necessary.

Universal Broker manages component start-up, execution, and termination. The Broker and its components have the ability to communicate service requests and status information between each other.


The Broker maintains a database of components that are active or have completed and waiting for restart or reconnection. The component information maintained by the Broker determines the current state of the component. This state information is required by the Broker to determine if a restart or reconnect request from a Manager is acceptable or not. The Broker's component information can be viewed with the [Universal Query](#) utility.

One piece of component information maintained by the Broker is the component's communication state. The communication state primarily determines what state the Server is in regarding its network connection with a Manager and the completion of the user process and its associated spooled data.

Communication State Values

The following table describes the communication state values.

- **Reconnect** column indicates whether or not a network reconnect request is valid.
- **Restart** column indicates whether or not a restart request is valid.

State	Reconnect	Restart	Description
STARTED	NO	NO	Server has started. If the Server is restartable, it is receiving the standard input file from the Manager and spooling it.
ESTABLISHED	NO	NO	Server and Manager are connected and processing normally. This is the most common state when all is well.
DISCONNECTED	YES	YES	Server is not connected to the Manager. This occurs when a network error has occurred, the Manager halted, or the Manager host halted. The Server is either executing with the Network Fault Tolerant protocol, is restartable, or both. <div style="background-color: #ffffcc; padding: 5px; border: 1px solid #ccc;"> <p> Note The Server cannot tell whether or not the Manager is still executing, since it cannot communicate with the Manager.</p> </div>
ORPHANED	NO	YES	Manager has terminated after sending a termination message to the Server to notify it of its termination. This state only occurs if the Server is restartable.
RECONNECTING	NO	NO	Server has received a reconnect request from the Manager to recover a lost network connection. This state should not remain long; only for the time it takes to re-establish the network connections.
RESTARTING	NO	NO	Server has received a restart request from the Manager. This state should not remain long; only for the time it takes to re-establish network connections.
PENDING	NO	YES	A restartable Server and its user process have completed. The user process standard output and error files are in the spool. A Manager has not been restarted to pick up the spooled files and user process exit status. The Server remains in this state until a Manager is restarted.
COMPLETED	NO	NO	Server and Manager have completed. All standard output and standard error files have been sent to the Manager and the user process's exit status.

Client Fault Tolerance - Universal Connector

Overview

The Client Fault Tolerance feature allows the Universal Connector client application to be shut down and restarted at a later time.

This functionality helps avoid problems that can result if the Universal Connector application terminates unexpectedly while processing an SAP job. In such an instance, the Client Fault Tolerance restart capability allows Universal Connector to reconnect to a running (or completed) job while preventing the unintentional start of a new instance of the original SAP job.

To achieve Client Fault Tolerance, Universal Connector must be able to associate the SAP jobs it defines and starts with a particular unit of work. In this context, a unit of work includes the Universal Connector client and SAP job instance.

To associate an SAP job with a particular unit of work, the user must be able to specify some identifying characteristic that is specific to that unit of work. The SAP system uniquely identifies job instances by a job name/job ID pair. Since the job name must be reusable and the job ID is assigned by the SAP system at the time of definition, Universal Connector must use an alternative job characteristic. This alternative job characteristic is the Universal Connector command Identifier.

Universal Connector references a particular unit of work by a job name/Command Identifier combination. The Universal Connector command identifier is tied to the SAP job by appending a Command ID Job Step to the SAP job associated with the Universal Connector command instance. The Command ID job step is required for identification purposes only. Therefore, the program used for the Command ID step is intended to add minimum overhead to the job. The Command ID used for the job is included in the definition of the Command ID job step.

Detailed Information

The following pages provide detailed information for Client Fault Tolerance - Universal Connector:

- [Modes](#)
- [Parameters](#)
- [Command ID Job Step](#)
- [Command Identifier](#)
- [Requesting Restart](#)

Client Fault Tolerance - Universal Connector - Modes

- [Overview](#)
- [Secure Client Fault Tolerance \(Secure CFT\) Mode](#)
- [Pre-XBP 2.0 Client Fault Tolerance \(CFT\) Mode](#)

Overview

Universal Connector supports two modes of client fault tolerance:

1. Secure Client Fault Tolerance (Secure CFT)
2. Pre-XBP 2.0 Client Fault Tolerance (CFT)

The mode of client fault tolerance to be used by Universal Connector is determined by the value of the `SECURE_CFT` option.

Valid values for this option are **yes** and **no**:

- **yes** will cause Universal Connector to use the Secure CFT mode.
- **no** will cause Universal Connector to use the original Pre-XBP 2.0 CFT mode.

The default value is **yes**.

Both modes of CFT follow the same basic process flow. When Universal Connector is requested to restart a particular command ID job, it queries the SAP system for all jobs with the specified job name. The list of jobs returned by the SAP system is scanned for a job that contains an appropriate Command ID Job Step. If found, Universal Connector will re-connect to the SAP job instance and satisfy the command line requirements.

Universal Connector is capable of restarting a command ID as long as the associated command ID job remains in the SAP system.

Secure Client Fault Tolerance (Secure CFT) Mode

This mode is an enhancement of the original implementation. The secure CFT mode requires XBP 2.0 to be installed on the SAP side of the Universal Connector connection. In this mode, an ABAP program step is used for the command ID job step.

Using an ABAP program step as the Command ID job step eliminates the security and ease of use drawbacks mentioned above for external program job steps.

- **Security**
The execution of ABAP programs and the resources required by them are secured by SAP authorization checks.
- **Ease of Use**
ABAP program job steps do not require a target host. They run on whichever application server the job runs on. Therefore, there are no target specific parameters required for secure CFT mode.

Pre-XBP 2.0 Client Fault Tolerance (CFT) Mode

This mode is the original implementation of client fault tolerance used prior to the release of XBP 2.0. Due to limitations in the XBP 1.0 interface, Universal Connector client fault tolerance on pre-XBP 2.0 SAP systems uses an external program step as the command ID job step.

Using an external program step as the command ID job step has the following security and ease of use drawbacks:

- **Security drawback**
Using an external program job step requires the SAP user ID to have authority to run external programs. This authority cannot be given lightly for the following reason: When running an external job step, the SAP system first performs an authorization check to see if the user ID has the right to run an external program. If so, the external program is run under the user ID of the user who started the SAP system
- **Ease of use drawback**
Using an external program job step requires a target host be specified for the external program to run on. This requires information about the SAP landscape that may not be readily available. Also, this presents the possibility of the Universal Connector job's parameters becoming out of sync with the SAP landscape.

Client Fault Tolerance - Universal Connector - Parameters

- [Client Fault Tolerance Target Host](#)
- [Client Fault Tolerance Command Prefix](#)
- [Secure Client Fault Tolerance Option](#)
- [Client Fault Tolerance ABAP Program](#)

Client Fault Tolerance Target Host

The client fault tolerance target host parameter is only required for pre-XBP 2.0 client fault tolerance mode. If the secure CFT mode is being used, the client fault tolerance target host parameter is ignored.

As part of an external program command ID job step definition, SAP requires a target host on which to run the external program (echo). Universal Connector provides the client fault tolerance target host parameter to specify the target host for the command ID job step.

The Client Fault Tolerance Target Host is specified with the [CFT_TARGET_HOST](#) option.

Client Fault Tolerance Command Prefix

The client fault tolerance command prefix parameter is only required for pre-XBP 2.0 client fault tolerance mode. If the secure CFT mode is being used, the client fault tolerance command prefix parameter is ignored.

The external program command ID job step has the potential to be run on any operating system reachable by the SAP system. The operating system that the Command ID Job Step runs on is that which exists on the host system specified by the Client Fault Tolerance Target Host parameter. Different operating systems may require commands to be called in different ways. Therefore, the Client Fault Tolerance Command Prefix parameter allows the user to specify the prefix necessary to run the echo command on the host system specified by the Client Fault Tolerance Target Host parameter.

For example, to run the echo command on a Windows system, the following command line entry would be required for an SAP external job step: **cmd /C echo**. Therefore, the Client Fault Tolerance Command Prefix for this system would be: **cmd /C**.

The Client Fault Tolerance Command Prefix parameter is specified with the [CFT_COMMAND_PREFIX](#) option.

Secure Client Fault Tolerance Option

The mode of client fault tolerance to be used by Universal Connector is determined by the value of the [SECURE_CFT](#) option.

Valid values for this option are **yes** and **no**:

- **yes** will cause Universal Connector to use the [Secure CFT](#) mode.
- **no** will cause Universal Connector to use the original [Pre-XBP 2.0 CFT](#) mode.

The default value is **yes**.

Client Fault Tolerance ABAP Program

The client fault tolerance ABAP program parameter is only required for secure CFT mode. If the pre-XBP 2.0 CFT mode is being used, the client fault tolerance ABAP program parameter is ignored.

The client fault tolerance ABAP program parameter is used to specify the ABAP program to use for the command ID job step. Any ABAP program can be specified. The Universal Connector internal default ABAP program to use is BTCTEST. BTCTEST is a standard SAP ABAP program that should be available on all SAP systems. It does no real processing and can be considered a dummy program that does not interfere with job processing and places little overhead on the system.

Client Fault Tolerance - Universal Connector - Command ID Job Step

- [Overview](#)
 - [Pre-XBP 2.0 CFT Mode](#)
 - [Secure CFT Mode](#)

Overview

Universal Connector creates Command ID jobs by appending a job step to the user's SAP job being defined to the system. This appended job step is the Universal Connector Command ID Job Step.

Pre-XBP 2.0 CFT Mode

In pre-XBP 2.0 CFT mode, the Universal Connector Command ID Job Step executes the external program echo. A string containing the command ID is inserted in the parameter field of the job step. The echo command is lightweight, does not interfere with the original job, and results in the command ID being printed to the joblog.

Secure CFT Mode

In secure CFT mode, the Universal Connector command ID job step executes an ABAP program. The ABAP program defined to the command id step is user configurable with the covetable parameter. Any ABAP program can be specified. The Universal Connector internal default ABAP program to use is BTCTEST. BTCTEST is a standard SAP ABAP program that should be available on all SAP systems. It does no real processing and can be considered a dummy program that does not interfere with job processing and places little overhead on the system.

Client Fault Tolerance - Universal Connector - Command Identifier

Command Identifier

Universal Connector requests client fault tolerance by providing a command identifier. The command identifier is specified on the command line with parameter **-cmdid**. The command ID/job name pair identifies the unit of work being executed.

The command ID option provides a command identifier that (paired with job name) uniquely identifies the SAP job on the SAP system. When a Universal Connector job is restarted, it must provide the same command ID identifying the SAP job with which it wants to reconnect.

Providing a unique command ID is not trivial. Many Universal Connector clients may be executing on many different hosts, all executing work on the same SAP system. It is possible for a Universal Connector client to start a restartable job from one host, terminate, and restart on a completely different host.

The command ID value can be any text value up to 245 characters in length. In practical terms, the character set and limits on command line length of the Universal Connector host may impose further restrictions on the value.

Client Fault Tolerance - Universal Connector - Requesting Restart

- [Requesting Restart](#)
- [Controlling Auto Restart](#)

Requesting Restart

When a restartable Universal Connector command is initiated, it is either an initial instance or a restarted instance of a command ID.

The [RESTART](#) option is specified on the command line with parameter **-restart**. RESTART specifies whether or not the Universal Connector command instance is requesting a restart of a previous command ID. Possible RESTART values are **yes**, **no**, or **auto**.

The **auto** value specifies that if there is no existing command ID job on the SAP system, consider this Universal Connector execution the first instance. If there is an existing command ID job, consider this a restart of the command ID. The **auto** value permits automatic restart by eliminating the need to modify the RESTART value for the initial instance and restarted instance.

It is important to note that when using the RESTART **auto** value, Universal Connector will not start a new instance of a job on the SAP system if a job matching the job name/command ID exists in the SAP system. Universal Connector will continue to reconnect to the existing SAP job.

Without considering the behavior resulting from the use of RESTART **auto**, it may be possible for one to assume that a job has been run multiple times when, in fact, Universal Connector has been reconnecting to the same job instance. Informational messages are printed by Universal Connector to standard error to indicate the reconnected status but, if the message level is not set to **info**, the messages will not be seen.

Controlling Auto Restart

Misunderstanding the auto restart behavior described above can potentially have serious consequences. For this reason, the [ALLOW_AUTO_RESTART](#) lets you allow or disallow the use of auto restart. You can make this specification in the Universal Connector configuration file and override it on the command line.

Sample Command Lines For Working With Client Fault Tolerance

Sample Command Lines For Working With Client Fault Tolerance

- [Working With Job Definition Files](#)
- [Working With Pre-defined SAP Jobs](#)

Working With Job Definition Files

- Initial Run of a Command ID Job
 - Pre-XBP 2.0 CFT Mode
 - Secure CFT Mode
 - Command Line Options
- Restart of a Command ID Job
 - Pre-XBP 2.0 CFT Mode
 - Secure CFT Mode
 - Command Line Options
- Run a Command ID Job Using Restart AUTO
 - Pre-XBP 2.0 CFT Mode
 - Secure CFT Mode
 - Command Line Options

Initial Run of a Command ID Job

The following examples will submit, start, and wait for the command ID job defined in job definition file **jobdef**. Because the RESTART option is set to **no**, Universal Connector will scan the SAP system to ensure that a command ID job with the same job name/command ID pair does not already exist on the system.

If a matching command ID job is found on the SAP system, Universal Connector will exit with an error code before performing the job submission.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cft_secure_cft no
      -cft_target_host pdf0196 -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -restart no
```



Note

The Client Fault Tolerance Command ID Prefix (`-cft_cmd_prefix`) is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix option can be specified in the configuration file and will never need to be specified on the command line. The same is true for the Client Fault Tolerance Target Host option (`-cft_target_host`). The secure CFT option (`-cft_secure_cft`) also would be set in the configuration file in most cases.

Secure CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cft_secure_cft yes
      -cft_abap BTCTEST -cmdid E8E8E80001 -restart no
```



Note

In secure CFT mode, the (`-cft_secure_cft`) and (`-cft_abap`) parameters would most likely be specified in the Universal Connector configuration file.

Command Line Options

Command line options used in these examples are:

Command Options	Description
<code>-userid</code>	Remote user ID with which to execute the command.

<code>-pwd</code>	Password for the user ID.
<code>-submit</code>	Defines a job to the SAP system.
<code>-start</code>	Starts the newly defined job.
<code>-wait</code>	Wait for the SAP job to complete processing.
<code>-cft_secure_cft</code>	Mode of client fault tolerance to be used for the command invocation.
<code>-cft_target_host</code>	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
<code>-cft_cmd_prefix</code>	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
<code>-cft_abap</code>	ABAP program to use for the command ID job step.
<code>-cmdid</code>	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
<code>-restart</code>	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.

Restart of a Command ID Job

In the following examples, Universal Connector is requested to restart command ID job E8E8E80001. Universal Connector will first parse the **jobdef** file to determine the jobname, and then scan the SAP system for a matching command ID job.

If a matching command ID job is found, Universal Connector reconnects to that job and satisfies the command line requirements. In this case, that means that if the job has not yet been started, it will be started, Universal Connector will wait for the job to complete (if it has not already), and the output will be returned.

If no matching command ID job is found, Universal Connector will terminate with an error code. Appropriate informational messages will be printed to standard error.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cmdid E8E8E80001 -restart yes
-cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cmdid E8E8E80001 -restart yes
-cft_secure_cft yes
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-----------------	-------------

-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Run a Command ID Job Using Restart AUTO

In the following examples, Universal Connector will first scan the SAP system to determine if a matching command ID job exists.

If no matching command ID job is found, Universal Connector considers this to be the initial instance of this command ID job and defines the new command ID job to the SAP system. If a matching command ID job is found, Universal Connector reconnects with the existing SAP job.

After determining if the command ID job is initial or a restart, Universal Connector satisfies the command line requirements.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cft_target_host pdf0196
-cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -restart auto -cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit jobdef -start -wait -cmdid E8E8E80001 -restart auto
-cft_secure_cft yes
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.

-submit	Defines a job to the SAP system.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.
-cft_target_host	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
-cft_cmd_prefix	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.

Working With Pre-defined SAP Jobs

- Initial Run of a Command ID Job
 - Pre-XBP 2.0 CFT Mode
 - Secure CFT Mode
 - Command Line Options
- Restart of a Command ID Job
 - Pre-XBP 2.0 CFT Mode
 - Secure CFT Mode
 - Command Line Options
- Run a Command ID Job Using Restart AUTO
 - Pre-XBP 2.0 CFT Mode
 - Secure CFT Mode
 - Command Line Options

Initial Run of a Command ID Job

The following examples will submit, start, and wait for the command ID job defined in the pre-existing SAP job with job name 'JOB_A' and job ID 19561301. Because the RESTART option is set to **no**, Universal Connector will scan the SAP system to ensure that a command ID job with the same job name/command ID pair does not already exist on the system.

If a matching command ID job is found on the SAP system, Universal Connector will exit with an error code before performing the job submission.

Note that the Client Fault Tolerance Command ID Prefix is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix parameter can be specified in the configuration file and will never need to be specified on the command line. The same may be true for the Client Fault Tolerance Target Host parameter.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
-cft_target_host pwwf0196 -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -restart no
-cft_secure_cft no
```



Note

The Client Fault Tolerance Command ID Prefix is set up for a Windows host. In many user environments, the Client Fault Tolerance Command ID Prefix parameter can be specified in the configuration file and will never need to be specified on the command line. The same may be true for the Client Fault Tolerance Target Host parameter.

Secure CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
-cft_secure_cft yes -cft_abap BTCTEST -cmdid E8E8E80001 -restart no
```



Note

In secure CFT mode, the **cft_secure_cft** and **cft_abap** parameters would most likely be specified in the Universal Connector configuration file.

Command Line Options

Command line options used in these examples are:

Command Options	Description

<code>-userid</code>	Remote user ID with which to execute the command.
<code>-pwd</code>	Password for the user ID.
<code>-submit</code>	Defines a job to the SAP system.
<code>-jobname</code>	Name of the SAP job.
<code>-jobid</code>	Job ID of the SAP job.
<code>-start</code>	Starts the newly defined job.
<code>-wait</code>	Wait for the SAP job to complete processing.
<code>-cft_target_host</code>	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
<code>-cft_cmd_prefix</code>	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
<code>-cmdid</code>	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
<code>-restart</code>	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
<code>-cft_secure_cft</code>	Mode of client fault tolerance to be used for the command invocation.

Restart of a Command ID Job

In the following example, Universal Connector is requested to restart command ID job E8E8E80001. Universal Connector will scan the SAP system for a matching command ID job.

If a matching command ID job is found, Universal Connector reconnects to that job and satisfies the command line requirements. In this case, that means that if the job has not yet been started, it will be started, Universal Connector will wait for the job to complete (if it hasn't already), and the output will be returned.

If no matching command ID job is found, Universal Connector will terminate with an error code. Appropriate informational messages will be printed to standard error.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
-cmdid E8E8E80001 -restart yes -cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
-cmdid E8E8E80001 -restart yes -cft_secure_cft yes
```

Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-jobname	Name of the SAP job.
-jobid	Job ID of the SAP job.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Run a Command ID Job Using Restart AUTO

In the following examples, Universal Connector will first scan the SAP system to determine if a matching command ID job exists.

If no matching command ID job is found, Universal Connector considers this to be the initial instance of this command ID job and defines the new command ID job to the SAP system. If a matching command ID job is found, Universal Connector reconnects with the existing SAP job.

After determining if the command ID job is initial or a restart, Universal Connector satisfies the command line requirements.

Pre-XBP 2.0 CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
-cft_target_host pdf0196 -cft_cmd_prefix "cmd /C" -cmdid E8E8E80001 -restart auto
-cft_secure_cft no
```

Secure CFT Mode

```
usap -userid bob -pwd secret -submit -jobname JOB_A -jobid 19561301 -start -wait
-cmdid E8E8E80001 -restart auto -cft_secure_cft yes
```


Command Line Options

Command line options used in these examples are:

Command Options	Description
-userid	Remote user ID with which to execute the command.
-pwd	Password for the user ID.
-submit	Defines a job to the SAP system.
-jobname	Name of the SAP job.
-jobid	Job ID of the SAP job.
-start	Starts the newly defined job.
-wait	Wait for the SAP job to complete processing.
-cft_target_host	In pre-XBP 2.0 CFT mode, the target host to use for the command ID job step when the command ID option is used.
-cft_cmd_prefix	In pre-XBP 2.0 CFT mode, the prefix command required for the operating system of the target host.
-cmdid	Identifier used to identify the unit of work represented by a USAP command and the associated SAP job.
-restart	Specification for whether or not this execution of Universal Connector is a restart of a previous client fault tolerant Universal Connector command.
-cft_secure_cft	Mode of client fault tolerance to be used for the command invocation.

Implementing Fault Tolerance - Examples

Implementing Fault Tolerance - Examples

- [Implementing Manager Fault Tolerance for Windows](#)

Implementing Manager Fault Tolerance for Windows

- [Implementing Manager Fault Tolerance for Windows](#)
 - [Command Line Options](#)
 - [Components](#)

Implementing Manager Fault Tolerance for Windows

The following figure illustrates how to activate manager fault tolerance. A unique command id is always required for manager fault tolerance.

```
ucmd -script script.file -host dallas -encryptedfile encrypted.file
      -managerft yes -cmdid uniquejobname -restart auto
```

The command is sent to a remote system, **dallas**, for execution. The output of the script is redirected back to the UCMD process. Additional command line options are read from the encrypted file, **encrypted.fil**e. Manager fault tolerance is turned on. A unique command ID, **uniquejobname**, is coded to identify the process.

Restart is detected automatically. If an executing or pending command ID exists, a reconnect is performed. If not, the process is started as if new.

Command Line Options

The command line options used in this example are:

Options	Description
-script	File from which to read a script file. The script file is sent to the remote system for execution.
-host	Directs the command to a computer with a host name of dallas .
-encryptedfile	File from which to read an encrypted command options file.
-managerft	Specification for whether or not the manager fault tolerant feature is used.
-cmdid	Unique command ID associated the unit of work.
-restart	Specification for whether or not the manager is requesting restart.

Components

[Universal Command Manager for Windows](#)

Opwise Universal Agent - Network Data Transmission

- [Introduction](#)
- [Network Protocols](#)
- [Detailed Information](#)

Introduction

Distributed systems, such as Opwise Universal Agent, communicate over data networks. All Opwise Universal Agent components communicate using the TCP/IP protocol; they do not use the UDP protocol for any product data communication over a network.

Network Protocols

Opwise Universal Agent can utilize either of two network protocols:

- **Secure Socket Layer Protocol**
Secure Socket Layer version 3 (**SSLv3**) provides the highest level of security available. SSL is a widely used and accepted network protocol for distributed software applications that are required to address all aspects of secure data transfer on private and public networks. All Opwise Universal Agent components (version 3.x and later) use **SSLv3**.
- **Universal V2 Protocol**
Universal V2 (version 2) legacy protocol, **UNVv2**, is provided for backward compatibility with Opwise Universal Agent (formerly Universal Products) versions earlier than 3.x, and when the SSL protocol resource utilization is considered too high. To ensure backward compatibility, this protocol is still supported by version 3.x components.

In addition to the network protocol used to transmit data, Opwise Universal Agent components use an application-layer protocol, [Universal Application Protocol](#), to exchange data messages.

Detailed Information

The following pages provide detailed information for Network Data Transmission:

- [SSL \(Secure Socket Layer\) Protocol](#)
- [Universal V2 Protocol](#)
- [Universal Application Protocol](#)
- [Network Data Transmission Tuning](#)
- [Network Data Transmission Configurable Options](#)

SSL (Secure Socket Layer) Protocol

- [Overview](#)
- [Data Privacy and Integrity](#)
 - [Encryption Algorithms](#)
 - [Message Digest Algorithms](#)
 - [Supported SSL Cipher Suites](#)
- [Peer Authentication](#)

Overview

Opwise Universal Agent implements the SSL protocol using the OpenSSL library or the IBM z/OS System SSL library, available on the z/OS operating system. The most recent SSL standard is version 3. A subsequent version was produced, changing the name to Transport Layer Security version 1 (**TLSv1**). **TLSv1** is the actual protocol used by Opwise Universal Agent. **TLSv1** is more commonly referred to simply as SSL and the term SSL is used throughout the rest of this documentation to mean **TLSv1**, unless otherwise noted.

The SSL protocol addresses the major challenges of communicating securely over a potentially insecure data network. This page discusses the issue of data privacy and integrity, and peer authentication.

Data Privacy and Integrity

People with sufficient technical knowledge and access to network resources can watch or capture data transmitting across the network. What they do with the data is up to them.

Data sent over the network that should remain private must be encrypted in a manner that unauthorized persons cannot determine what the original data contained regardless of their level of expertise, access to network resources, amount of data captured, and amount of time they have. The only party that should be able to read the data is the intended recipient.

As data is transmitted over the network, it passes through media and hardware of unknown quality that may erroneously change bits of data without warning. Additionally, although data may be encrypted, there is nothing stopping a malicious person from changing the data while it is transmitted over the network. The changed data may or may not be detected by the recipient depending on what changed and how it is processed. It may be accepted as valid data, but the information it represents is now erroneous.

Data integrity must be protected from errors in transmission and malicious users. Data integrity checks insure that what was sent is exactly what is received by the recipient. Without integrity checks, there is no guarantee.

Encryption Algorithms

Encryption algorithms are used to encrypt data into an unreadable format. The encryption process is computationally expensive. There are a variety of encryption algorithms some of which perform better than others. Some algorithms offer a higher level of security than others. Typically, the higher level of security requires more computational resources.

Message Digest Algorithms

Message digest algorithms are used to produce a Message Authentication Code (MAC) that uniquely identifies a block of data. The sender computes a MAC for the data being sent based on a shared secret key the sender and receiver hold. The sender sends the data and the MAC to the receiver. The receiver computes a new MAC for the received data based on the shared secret key. If the two MACs are the same, data integrity is maintained, else the data is rejected as it has been modified.

Message digest algorithms are often referred to as MACs and can be used synonymously in most contexts.

Supported SSL Cipher Suites

The SSL standard defines a set of encryption and message digest algorithms, referred to as cipher suites, that insure data privacy and data integrity. Cipher suites pair encryption algorithms with appropriate message digest algorithms. The two algorithms cannot be specified individually.

Opwise Universal Agent supports a subset of the complete SSL cipher suites defined by the standard. The cipher suite name is formatted as an encryption algorithm abbreviation followed by the message digest algorithm abbreviation.

The following table identifies the supported SSL cipher suites.

Cipher Suite Name	Description
RC4-SHA	128-bit RC4 encryption with SHA-1 message digest

RC4-MD5	128-bit RC4 encryption with MD5 message digest
AES256-SHA	256-bit AES encryption with SHA-1 message digest
AES128-SHA	128-bit AES encryption with SHA-1 message digest
DES-CBC3-SHA	128-bit Triple-DES encryption with SHA-1 message digest
DES-CBC-SHA	128-bit DES encryption with SHA-1 message digest
NULL-SHA	No encryption with SHA-1 message digest
NULL-MD5	No encryption with MD5 message digest

Opwise Universal Agent supports one additional cipher suite name that is not part of the SSL protocol. The NULL-NULL cipher suite turns SSL off completely and instead uses the [Universal V2 Protocol|Universal V2 Protocol] (**UNVv2**).

Selecting an SSL Cipher Suite

When two Opwise Universal Agent components (for example, a UEM Manager and a UEM Server) first connect, they perform an SSL handshake that negotiates the cipher suite to use for the session. The Manager presents a list of cipher suites (in descending order of preference) that it would like to use. This is compared against a list of ciphers that the Server supports. The first cipher suite in common is the one used for the session.

Lists of cipher suites are helpful where a distributed software solution may cross many organizational and application boundaries, each with its own security requirements. Instead of having to choose one cipher suite for all distributed components, the software components can be configured with their own list of acceptable cipher suites based on their local security requirements.

When a high level of security is required, the higher CPU consuming cipher suite is justified. When lower level of security is acceptable, a lower CPU consuming cipher suite can be used. As long as the Manager has both cipher suites in its list, it can negotiate either cipher suite with servers of different security levels.

Peer Authentication

When communicating with a party across a data network, how do you insure that the party you are communicating with (your peer) is who you believe? A common form of network attack is a malicious user representing themselves as another user or host.

Peer authentication insures that the peer is truly who they identify themselves as. Peer authentication applies to users, computer programs and hardware systems.

SSL uses X.509 certificates and public and private keys to identify an entity. An entity may be a person, a program, or a system. A complete description of X.509 certificates is beyond the scope of this documentation. [X.509 Certificates](#) provides an overview to help get the reader oriented to the concepts, terminology and benefits.

For additional details, the following web site is recommended:

<http://www.faqs.org/rfcs/rfc3280.html>

Universal V2 Protocol

Universal V2 Protocol

The Universal V2 protocol, **UNVv2**, is a proprietary protocol that securely and efficiently transports data across data networks. **UNVv2** was the only supported protocol in Opswise Universal Agent (formerly Universal Products) prior to version 3 and will be available in future versions.

UNVv2 addresses data privacy and integrity. It does not address peer authentication.

Data Privacy and Integrity

Data privacy is insured with data encryption algorithms. **UNVv2** utilizes 128-bit RC4 encryption for all data encryption.

Data integrity is insured with message digest algorithms. **UNVv2** utilizes 128-bit MD5 MACs for data integrity. **UNVv2** referred to data integrity as data authentication.

Encryption and integrity may be enabled and disabled on an individual bases.

Encryption keys are generated using a proprietary key agreement algorithm. A new key is created for each and every network session.

Universal Application Protocol

- Universal Application Protocol
- Low-Overhead
- Secure
- Extensible

Universal Application Protocol

Opwise Universal Agent components use an application-layer protocol to exchange data messages. The protocol has the following characteristics:

- Low-Overhead
- Secure
- Extensible
- Configurable Options

The following information refers to two categories of data transmitted by Opwise Universal Agent:

1. Control data (or messages) consists of messages generated by Opwise Universal Agent components in order to communicate with each other. The user of the product has no access to the control data itself.
2. Application data (or messages) consists of data that is transmitted as part of the requested work being executed. For example, standard input and output data of jobs Universal Command executes. The data is created by the job and read or written by Universal Command on behalf of the job.

Low-Overhead

The protocol is lightweight, in order to minimize its use of network bandwidth. The product provides application data compression options, which reduces the amount of network data even further.

There are two possible compression methods:

1. **ZLIB** method offers the highest compression ratios with highest CPU utilization.
2. **HASP** method offers the lowest compression ratios with lowest CPU utilization.



Note

Control data is not compressed. Compression options are available for application data only.

Secure

When used by Opwise Universal Agent Managers prior to version 3.x, and when communicating with Opwise Universal Agent Servers that force encryption on, the UNVv2 protocol is secure.

All control data exchanged between Opwise Universal Agent components are encrypted with a unique session key and contain a MAC. The encryption prevents anyone from analyzing the message data and attempting to circumvent product and customer policies. Each session uses a different encryption key to prevent "play back" types of network attacks, where messages captured from a previous session are replayed in a new session. This applies to both network protocols: [SSL](#) and [UNVv2](#).

In versions prior to Opwise Universal Agent, the security features used in the control messages are not optional. They cannot be turned off. The security features are optional for application data sent over the network.

Starting with Opwise Universal Agent, the UNVv2 protocol is used only when SSL is disabled on the control session by specifying the NULL-NUL cipher suite. In this case, the UNVv2 encryption or MACs are not used for control messages.

As of Opwise Universal Agent, the SSL protocol must be used if data privacy and integrity is required for control messages. For this reason, UNVv2 should only be used when the resource utilization of SSL is considered too high and data privacy is not required. It is Stonebranch's recommendation that SSL should be used if at all possible to insure data privacy and data integrity.

Backward compatibility is still maintained with Opwise Universal Agent (formerly Universal Products) versions prior to 3.x such that encryption and MAC's are still utilized for the control session.

Extensible

The message protocol used between the Opswise Universal Agent components is extensible. New message fields can be added with each new release without creating product component incompatibilities. This permits different component versions to communication with each other with no problems. This is a very important feature for distributed systems, since it is near impossible to upgrade hundreds of servers simultaneously.

New encryption and compression algorithms can be added in future releases without losing backward compatibility with older releases. After a network connection is made, connection options are negotiated between the two Opswise Universal Agent programs. The options negotiated include which encryption and compression algorithms are used for the session. Only algorithms that both programs implement are chosen in the negotiation process. The negotiation process permits two different program versions to communicate.

Network Data Transmission Tuning

- [Overview](#)
- [Bandwidth Delay Product](#)
- [TCP High Performance Extensions](#)
- [TCP Buffers](#)
 - [TCP Buffer Configuration](#)

Overview

TCP/IP tuning is considered an advanced subject. The background information necessary for a complete understanding of the subject is beyond the scope of this document. If misconfigured, the product configuration options for data transmission tuning can have a negative impact on the network performance of product components as well as on the TCP/IP network.

Product components that support bulk data transfers benefit the most from the network tuning described in this section. For example, Universal Data Mover provides the ability to transfer very large files between systems, so it would be a candidate for network tuning.

The network tuning technique described in this section addresses the problem of transferring data over certain types of transmission links, primarily large bandwidth, high latency transmission links. Today's transmission links can exceed 1 Gbit/s with a round trip time of 50 ms or more. The default TCP/IP buffers are not suitable for optimized data transmission over such links.

Bandwidth Delay Product

The bandwidth delay product (BDP) measures the amount of data that a transmission link holds. The BDP is used to help tune specific TCP/IP configuration options.

The BDP is calculated as the product of the maximum bandwidth and the round trip time (RTT) of the transmission link. BDP is expressed in bytes. The maximum bandwidth of a link is limited by the slowest part, or bottleneck, of the network route. As an example, consider a network route that starts on a server with a 100 Mbit/s network interface card that is connected to a 1 Gbit/s network and ends on a server with a 1 Gbit/s network interface card. The maximum bandwidth is the slowest part of the route, which is the 100 Mbit/s network interface card. There is no reliable way to measure bandwidth in all cases. A knowledge of the network topology is required to know the slowest part of a network route. The RTT is measurable using the ping command. Simply use the ping command to "ping" the remote destination and it will report the RTT in milliseconds.

The BDP formula is shown below.

$$(B / 8) * (T / 1000)$$

where, B is the bandwidth measured in bits per second and T is the RTT measured in milliseconds.

As an example, if the maximum bandwidth is 1 Gbit/s and the RTT is 60 ms, the BDP is calculated as

$$(1,000,000,000 / 8) * (60 / 1000) = 7,500,000 \text{ bytes} = 7.5 \text{ MB}$$

TCP High Performance Extensions

Originally, TCP/IP was not optimized for transmission links with a high bandwidth delay product (BDP). [RFC 1323 TCP Extensions for High Performance](#) introduced changes to the TCP protocol to improve performance over high BDP links. RFC 1323 includes a number of TCP changes, but the most relevant one for this discussion is the window scaling option.

The TCP receive window size is negotiated by the TCP implementations during the three-way handshake when the connection is opened. The window specifies the amount of buffer space the receiving TCP has available for data. The TCP sender does not send any more data than the receiver's available window. The TCP window is a form of flow control to prevent the sender from sending more data than the receiver has buffer space available.

The TCP receive window is defined in the TCP header as a 16 bit field, so the maximum window size is 65 KiB. For a transmission link with a large BDP, this is only a fraction of the amount of data the transmission link will hold. Consequentially, the transmission link will never fill to capacity and maximum bandwidth never achieved. RFC 1323 added the window scaling option so that a larger TCP window can be negotiated. The window scaling option makes the TCP window size effectively a 32-bit value, however RFC 1323 does limit it to 1 GiB. The TCP implementations on both sides of the socket connection must support window scaling for it to be used.

TCP Buffers

The TCP receive window used by the TCP on the receiving end of a connection is typically determined from the size of the TCP receive buffer used for the connection. The default TCP receive buffer can typically be configured as part of the TCP configuration. However the default is not typically large enough for high BDP transmission links. The TCP socket API provides an interface for the application to request specific TCP receive and send buffer sizes. The application can request any buffer size and TCP will determine what size it actually uses based on its configuration limits. If TCP on both ends of the socket connection support RFC 1312 window scaling, the TCP window may be as large as 1 GiB if the TCP configuration permits.

TCP Buffer Configuration

In general, the optimum TCP buffer size matches the BDP for the transmission link. However, TCP buffers are maintained by TCP in virtual storage. Very large buffer sizes may actually reduce transmission rates if the virtual storage requirements exceed the system memory capabilities.

Some product components provide configuration options to specify the TCP send and receive buffer sizes. Configuration options `TCP_RECV_BUFFER` and `TCP_SEND_BUFFER` specify the TCP receive and send buffer sizes, respectively. The product components on both ends of the TCP socket connection must be configured using the `TCP_RECV_BUFFER` and `TCP_SEND_BUFFER` options. Product components typically consist of a Manager component (such as UDM Manager) and a Server component (such as UDM Server). The actual connection between the Manager and Server is established first with the Universal Broker component. A Manager component first establishes a socket connection with the Broker which then starts the Server component and passes the socket connection to the Server. Consequentially, the Universal Broker will always require TCP buffer configuration changes in order to tune network performance of product components.

Network Data Transmission Configurable Options

- Configurable Options
 - CODE_PAGE
 - CTL_SSL_CIPHER_LIST
 - DATA_AUTHENTICATION
 - DATA_COMPRESSION
 - DATA_ENCRYPTION
 - DATA_SSL_CIPHER_LIST
 - DEFAULT_CIPHER
 - ENCRYPT_CONTROL_SESSION
 - KEEPALIVE_INTERVAL
 - NETWORK_DELAY
 - SIO_MODE

Configurable Options

The network protocol can be configured in ways that affect compression, encryption, code pages, and network delays.

The following configuration options are available on many Opswise Universal Agent components:

CODE_PAGE

The CODE_PAGE option specifies the code page translation table used to translate network data from and to the local code page for the system on which the program is executing.

A codepage table is text file that contain a two-column table. The table maps local single byte character codes to two-byte UNICODE character codes.

Code pages are located in the product National Language Support (NLS) directory or library. New code pages may be created and added to the NLS directory or library. The CODE_PAGE option value is simply the name of the code page file without any file name extension if present.

CTL_SSL_CIPHER_LIST

The CTL_SSL_CIPHER_LIST option specifies one or more SSL cipher suites that are acceptable to use for network communications on the control session, which is used for component internal communication.

The SSL protocol uses cipher suites to specify the combination of encryption and message digest algorithms used for a session. An ordered list of acceptable cipher suites can be specified in a most-to-least order of preference.

An example cipher suite list is RC4-MD5,RC4-SHA,AES128-SHA. The RC4-MD5 cipher suite is the most preferred and AES128-SHA is the least preferred.

When two Opswise Universal Agent components (Manager and a Server) first connect, they perform an SSL handshake that negotiates the cipher suite to use for the session. The Manager presents a list of cipher suites (in descending order of preference) that it would like to use. This is compared against a list of ciphers that the Server supports. The first cipher suite in common is the one used for the session.

DATA_AUTHENTICATION

The DATA_AUTHENTICATION option specifies whether or not the network data is authenticated. Data authentication verifies that the data did not change from the point it was sent to the point it was received.

Data authentication also is referred to as a data integrity in this document.

Data authentication occurs for each message sent over the network. If a message fails authentication, the network session is terminated and both programs end with an error.

The DATA_AUTHENTICATION option is applicable to the **UNVv2** protocol only. SSL always performs authentication.

DATA_COMPRESSION

The DATA_COMPRESSION option specifies that network data be compressed.

Compression attempts to reduce the amount of data to a form that can be decompressed to its original form. The compression ratio is the original size divided by the compressed size. The compression ratio value will depend on the type of data. Some data compress better than others.

Two methods of compression are available:

1. ZLIB method provides the highest compression ratio with the highest use of CPU
2. HASP method provides the lowest compression ratio with the lowest use of CPU.

Whether or not compression is used and which compression method is used depends on several items:

- Network bandwidth. If network bandwidth is small, compression may be worth the cost in CPU.
- CPU resources. If CPU is limited, the CPU cost may not be worth the reduced bandwidth usage.
- Data compression ratio. If the data does not compress well, it is probably not worth CPU cost. If the data ratio is high, the CPU cost may be worth it.

DATA_ENCRYPTION

The DATA_ENCRYPTION option specifies whether or not network data is encrypted.

Encryption translates data into a format that prevents the original data from being determined. Decryption translates encrypted data back into its original form.

The type of encryption performed depends on the network protocol being used, SSL or **UNVv2**.

Data encryption does increase CPU usage. Whether or not encryption is used depends on the sensitivity of the data and the security of the two host systems and the data network between the hosts.

DATA_SSL_CIPHER_LIST

The DATA_SSL_CIPHER_LIST option specifies one or more SSL cipher suites that are acceptable to use for network communications on the data session, which is used for standard I/O file transmission.

(See [CTL_SSL_CIPHER_LIST](#).)

DEFAULT_CIPHER

The DEFAULT_CIPHER option specifies the SSL cipher suite to use (since SSL protocol requires a cipher suite) if the DATA_ENCRYPTION option is set to **no**. The default DEFAULT_CIPHER is NULL-MD5 (no encryption, MD5 message digest).

All SSL cipher suites have a message digest for good reasons. The message digest ensures that the data sent are the data received. Without a message digest, it is possible for bits of the data packet to get changed without being noticed.

ENCRYPT_CONTROL_SESSION

The ENCRYPT_CONTROL_SESSION option is a server-only option that enforces encryption on the control session. When the option is set to a value of **no**, the server will accept a control session protocol without encryption and message authentication codes (MACs). The default is **yes**.

Starting with Opwise Universal Agent, a manager can request that the UNVv2 protocol be used without encryption or MACs. Considering that host systems may require differing security policies, this option allows for each server to be configured appropriately based on its security policy.

KEEPALIVE_INTERVAL

The KEEPALIVE_INTERVAL option specifies how often, in seconds, a keepalive message (also commonly known as a heartbeat message) is sent between a manager and server.

A keepalive message ensures that the network and both programs are operating normally. Without a keepalive message, error conditions can arise that place one or both programs in an infinite wait.

A keepalive message is sent from the server to the manager. If the server does not receive a keepalive acknowledgement from the manager in a certain period of time (calculated as the maximum of 2 x [NETWORK_DELAY](#) or the KEEPALIVE_INTERVAL), the server considers the manager or network as unusable.

How the server processes a keepalive time-out depends on what fault tolerant features are being used. If no fault tolerant features are being used, the server ends with an error. The manager expects to receive a keepalive message in a certain period of time (calculated as the KEEPALIVE_INTERVAL + 2 x [NETWORK_DELAY](#)).

NETWORK_DELAY

The NETWORK_DELAY option provides the ability to fine tune Opwise Universal Agent network protocol. When a data packet is sent over a TCP/IP network, the time it takes to reach the other end depends on many factors, such as, network congestion, network bandwidth, and the network media type. If the packet is lost before reaching the other end, the other end may wait indefinitely for the expected data.

In order to prevent this situation, Opwise Universal Agent components time out waiting for a packet to arrive in a specified period of time. The

delay option specifies this period of time.

NETWORK_DELAY specifies the maximum acceptable delay in transmitting data between two programs. Should a data transmission take longer than the specified delay, the operation ends with a time out error. Opswise Universal Agent components will consider a time out error as a network fault.

The default NETWORK_DELAY value is 120 seconds. This value is reasonable for most networks and operational characteristics. If the value is too small, false network time outs could occur. If the value is too large, programs will wait a long period of time before reporting a time out problem.

SIO_MODE

The SIO_MODE option specifies whether the data transmitted over the network is processed as text data or binary data.

Text data is translated between the remote and local code pages. Additionally, end of line representations are converted

Text translation operates in two modes: direct and UCS. The default is direct. The direct translation mode exchanges code pages between Opswise Universal Agent components to build direct translation tables. Direct translation is the fastest translation method when a significant amount (greater than 10K) of text data is transmitted. The code page exchange increases the amount of data sent over the network as part of the network connection negotiation.

UCS translation does not require the exchange of code pages. For transactions that have little text data transmission, this is the fastest.

Binary data is transmitted without any data translation.