



NetWeave Integrated Solutions, Inc.

# NetWeave Configuration Guide

---

*User's Guide for Version 2.0 January 2008*



[www.netweave.com](http://www.netweave.com)

Copyright © 2002-2008 NetWeave Integrated Solutions, Inc.. All rights reserved.

Netweave is a registered trademark of Netweave Integrated Solutions, Inc.

Windows is a registered trademark of Microsoft Corporation.

CICS, MVS, and MQSeries are registered trademarks of the IBM Corporation.

UNIX is a registered trademark of The Open Group.

Tandem, Guardian, VMS, and OpenVMS are registered trademarks of Hewlett-Packard.

All other trademarks are noted in the text and are the property of their respective owners.

.

# Table of Contents

<b>INTRODUCTION .....</b>	<b>1</b>
What Is NetWeave?.....	1
The NetWeave Documentation Suite .....	2
Installing NetWeave .....	3
Object Code vs. Source Code .....	3
ANSI C .....	3
The NetWeave Components.....	3
The Architecture of the NetWeave Library .....	6
<b>CREATING AND EDITING INI FILES .....</b>	<b>7</b>
The INI File.....	7
The INI File Groups .....	8
The [LICENSE_GROUP] Group.....	8
The [TRACES] Group.....	8
Message Log Facility .....	12
The Root Group.....	13
Naming Services .....	14
Logical Names and Aliases .....	14
Message Routing.....	14
Load Balancing.....	15
Load Balancing Example .....	16
Configuration Information for Applications.....	17
Substitution Strings .....	18
<b>CONFIGURING SESSIONS AND SERVICES.....</b>	<b>19</b>
Data Conversion.....	19
Data Compression .....	20
Data Compression Parameters.....	20
Data Compression Examples.....	21
Algorithm Technical Information.....	21
NetWeave Security Services .....	24
Authentication By Name and Password.....	24
Authentication Details By Platform .....	25
Authentication By Challenge-Response.....	26
Encryption .....	27
<b>THE NETWEAVE COMMUNICATIONS STACK.....</b>	<b>31</b>
Using Heartbeats to Monitor a Communications Stack .....	31
Using Capping and Pooling to Control How Connections are Used.....	32

Connection Timeouts .....	34
<b>THE PROTOCOL LAYER .....</b>	<b>35</b>
TCP/IP Protocol .....	35
BROADCAST Protocol .....	37
Parameters to Control the Rate of Broadcasts .....	37
Parameters to Control Accurate Delivery of Broadcasts.....	38
Broadcast Parameters for the [LEVEL2] Group .....	38
Broadcast Parameters for a Protocol Group .....	40
Sample Files for Broadcasting.....	43
Sender's Sample INI File .....	44
Broadcaster's INI File .....	44
Reader's INI File .....	45
DOLLAR_RECV Protocol (Tandem).....	46
PATHSEND Protocol (Tandem).....	47
QACCESS Protocol (IBM/CICS) .....	48
Dual-Rail Routing .....	50
<b>MISCELLANEOUS CONFIGURATION ISSUES.....</b>	<b>51</b>
Compatibility With Older Versions of NetWeave .....	51
SQL Server for Tandem .....	51
The NetWeave Dispatcher.....	52
The [Blocking] Group .....	52
The [FILE_COPY] Group.....	53
The [RPC_CLIENT] Group .....	53
The [RPC_SERVER] Group.....	53
The [LEVEL4] Group .....	53
The [NWDS_QUEUE_CONTROL] Group.....	54
Setting the File Type .....	54
<b>HIERARCHY OF INI FILES.....</b>	<b>55</b>
Dynamic INI Files .....	56
Starting INI Server .....	57
A Sample INI File for INI Server.....	57
A Sample Boot INI for an Application.....	58
A Sample Common INI File.....	60
Runtime Parameters That You Can Change.....	61
When Changes Take Effect .....	61
Chains of INI Files .....	62
Chained File Example .....	62
Chained Files and Load Balancing.....	63

The Dynamic INI Management Interface.....	65
Starting INI Manager.....	65
Example 1.....	65
Example 2.....	66
<b>MESSAGE AND ERROR LOGGING CONSIDERATIONS.....</b>	<b>68</b>
Basic Error Tracing.....	68
Application Message Logging.....	69
Platform-Specific Logging.....	69
Windows NT.....	70
MVS/CICS.....	71
Tandem.....	71
UNIX.....	71
<b>SUPPORTED PLATFORMS.....</b>	<b>72</b>
<b>SAMPLE INI FILES.....</b>	<b>73</b>
Simple Communication Between Two Processes.....	73
NetWeave FIFO Queues.....	74
Producer's INI File.....	74
Consumer's INI File.....	75
<b>GLOSSARY.....</b>	<b>77</b>

## History of Revisions

Initial Release:	October 1995
Revision 1:	October 1996
Revision 2:	December 1997
Revision 3:	September 1999
Revision 4:	November 1999
Revision 5:	February 2000
Revision 6:	January 2001
Revision 7:	January 2003



## Introduction

### What Is NetWeave?

The **NetWeave Distributed Services (NWDS)** product provides the interoperability services required to build distributed applications on a heterogeneous network of hardware platforms, applications, and databases. The NetWeave library of remote services (the NetWeave API) provides software developers with messaging and data query and update capabilities wherever the data resides and in whatever form it exists.

This manual explains how to install and configure a NetWeave system for a customer's particular environment, with special attention to those services and support files that would normally be managed by someone other than an applications programmer. In this manual you will learn how to:

- Configure the interface to the communications layer
- Define an application's architectural components in a heterogeneous computing environment
- Manage the operational details required to make NetWeave available to the applications programmer

The NetWeave configuration information is stored in text files called *INI files*. Once you configure NetWeave for an application, most of the settings in your INI files will not change. For systems and operations staff, this means that it is easy to integrate NetWeave startup and shutdown procedures with other standard system procedures. Although most configuration information tends to be quite static, NetWeave also provides what are called dynamic INI files that allow you to make configuration changes while an application is running.

Before attempting specific NetWeave tasks such as configuring INI files, please read the rest of the introductory information in this chapter for an overview of the NetWeave architecture and installation requirements.



## The NetWeave Documentation Suite

The table below lists the documentation for the NetWeave product. You can download these documents from the NetWeave ftp site by connecting to <ftp://www.netweave.com/middleware/doc/200/MSWord> (for Microsoft Word format) or <ftp://www.netweave.com/middleware/doc/200/PDF> (for PDF format).

Document	Description	Audience
Configuration Guide	Explains how to install and configure a NetWeave system for a customer's particular environment.	Systems programmers who maintain the communications layer on which NetWeave rests. Operations personnel who start and shut down NetWeave processes in a distributed environment. Designers and managers who configure the applications that use NetWeave.
Programmer's Guide	Explains how to use NetWeave functions to meet the requirements of distributed systems. Illustrates working sample programs.	Analysts who design and programmers who build applications in a distributed computing environment.
API Guide	Describes the NetWeave API's function calls for client-transaction applications, messaging services, and client-database applications.	Programmers who create and maintain NetWeave applications on any system.
IBM/CICS Configuration Supplement	Explains and illustrates features of NetWeave that are unique to the IBM/CICS environment.	System managers and analysts who must install NetWeave in the IBM/CICS environment.
NetWeave Print Process for Tandem	Describes how to install and use this printer driver to transfer spooled output from a Tandem system to any computer on which NetWeave is installed.	Tandem system managers.
Enhancements	Lists enhancements and significant bug fixes by release version.	Project managers and programmers who plan, build and maintain NetWeave applications.
Performance Tester	Explains how to use the Performance Test program to simulate and measure a variety of NetWeave application configurations.	Project managers and programmers who plan, build and maintain NetWeave applications.





## Installing NetWeave

### Object Code vs. Source Code

The NetWeave installation team may not possess the unique combination of hardware platforms, operating systems, or communications libraries that a customer intends to use. When we do, we install object code and link it with the customer's libraries. When we don't, we bring source code to the customer's site temporarily and compile NetWeave on the customer's computers.

### ANSI C

NetWeave is written in ANSI C. Normally we supply all code as libraries and executable programs. If NetWeave cannot furnish the entire product as object code, you must have an ANSI C compiler on your computer. Even when NetWeave can provide object code, you must have the C runtime library on your system.

## The NetWeave Components

NetWeave has four main components:

Component	Description
NetWeave Library	A library bound with the user's application that provides NWDS functions to the application.
NetWeave Agent	A standalone process that performs functions on behalf of the user's application.
INI Server	A standalone process that supplies information to an application from dynamic INI files.
INI Manager	A standalone process that controls INI Server operations. INI Manager also supports dynamic INI files.

The NetWeave API is a collection of functions that are distributed as a library appropriate to a specific computing platform. A NetWeave-enabled application is a program that calls one or more of the NetWeave API functions. The NetWeave Agent, INI Server, and INI Manager are NetWeave-enabled applications that use the NetWeave library to interoperate. The remainder of this section explains the role of the library and the Agent in communicating and coordinating program activities.

A NetWeave-enabled application (also called a *NetWeave application*) on one platform can use the NetWeave API to communicate with and request services from NetWeave applications and NetWeave Agents on other platforms, even when the platforms are connected with different communications protocols.

NetWeave provides interfaces to the following protocols:

- TCP/IP
- UDP/IP
- Dual-rail, redundant communication over parallel links
- HTTP
- A proprietary IPC mechanism for Tandem's Guardian operating system
- A proprietary IPC mechanism for IBM/CICS

In Figure 1, a NetWeave application (the one in the middle) on one platform can communicate with and request services from NetWeave applications on other platforms, even when the platforms communicate using different protocols.

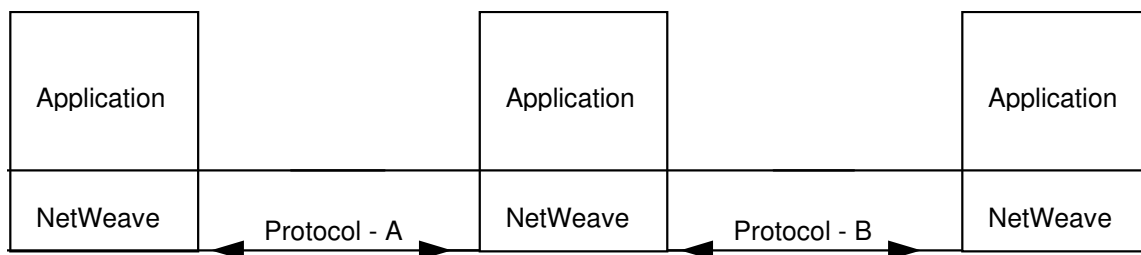


Figure 1. Communications among NetWeave applications

Because the NetWeave Agent on one computer may perform services for client applications located on the same or different computers, the Agent may play several different roles in your application's architecture. In the role of *protocol converter*, the Agent allows two applications that use different communications protocols to exchange messages. In its role as *NetWeave Server*, the Agent may perform database or broadcast services – most commonly by supporting NetWeave FIFO queues in support of workflow architectures.

To access a file on a remote machine, a NetWeave application must use the NetWeave Agent installed on the remote platform to access the requested file. To access files on its local system, an application doesn't need a NetWeave Agent; it uses the NetWeave API. If the API is set up to handle file I/O, then even if a file no longer resides on the local system, you won't have to update the application. The only change required in the application's configuration file is the entry for the new filename.

**NOTE:** Because the NetWeave Agent is a standalone process (it has no API), it can be accessed only through the NetWeave API.

Figure 2 on the next page illustrates the role of the NetWeave Agent.

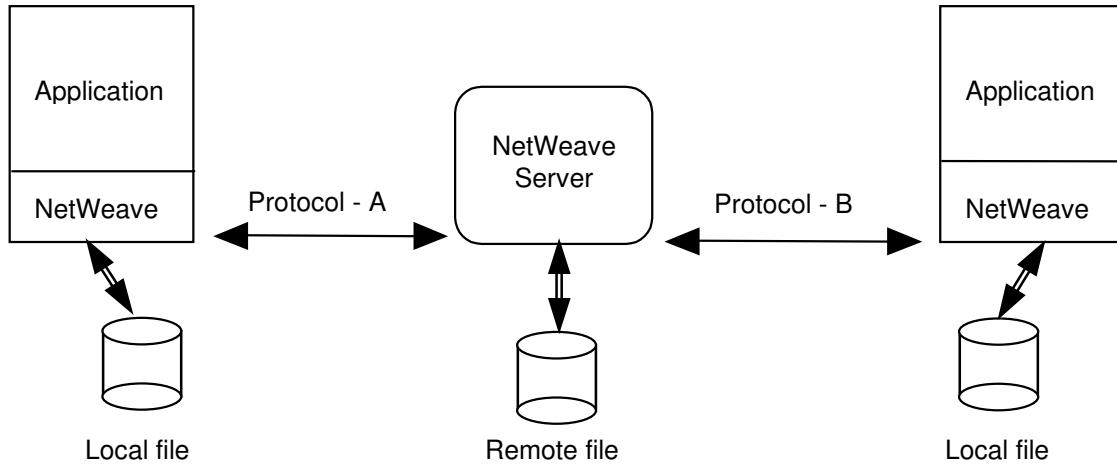


Figure 2. The role of the NetWeave Agent

The table below compares a NetWeave application and a NetWeave Agent:

Module	Attribute
NetWeave Application and Agent	A NetWeave application can communicate with other NetWeave applications and/or NetWeave Agents; a NetWeave Agent can communicate with other NetWeave Agents and/or NetWeave applications. A NetWeave library and Agent installed on the same platform share over 80% of the same code.
NetWeave Agent only	Processes NetWeave file I/O requests from a remote NetWeave application. Reads and writes to NetWeave FIFO queues. Routes messages from one NetWeave application to another. Acts as a protocol converter. Sends and receives broadcast messages.

Because the Agent is a NetWeave-enabled application that uses the NetWeave library, you configure it just you would any other NetWeave-enabled application.



## The Architecture of the NetWeave Library

There are two ways to look at configuration parameters:

- Vertically, as functionally related collections of API functions. You can configure either interprocess communication (IPC) or FIFOs.
- Horizontally, by concentrating on sessions that may contain multiple, functionally unrelated API calls. The library is a series of layers, each of which encompasses specific functional responsibilities. These layers interoperate through fixed interfaces, allowing you to change details within one layer without affecting the operation of another layer.

Because many of the configuration parameters apply to both IPC and FIFOs, our documentation takes the second (horizontal) approach. From this perspective, the library consists of three principal architectural layers:

1. The session or *services layer* is the interface between the user's application and the transport layer. Some of the more important services are authentication and encryption, message compression, and local file services.
2. The transport layer or *communications stack* provides a generic message-passing mechanism that the services use for their distributed functions. The communications layer shields the services layer from the idiosyncrasies of the protocols.
3. The *protocol layer* provides the direct interfaces between NetWeave and platform-specific function calls.

Most of the user-provided configuration parameters affect the services and the protocol layers. Although some parameters do affect the communications stack, usually you will not need to change their default values.



## Creating and Editing INI Files

### The INI File

An INI file is a text file that contains case-sensitive lists of *parameters* arranged into functional *groups*. Each group consists of a group name in square brackets, followed by a list of one or more parameters (keyword-value pairs separated by the equal sign). Although a parameter is defined as a keyword plus its value, we occasionally refer to the keyword alone as the parameter. A *comment* in an INI file is preceded by an asterisk (\*) in the leftmost column.

**NOTE:** The syntax and parameter information in this section applies to all layers of the NetWeave library. Please keep in mind that INI file information is case-sensitive; thus names such as [Group1] and [GROUP1] are not the same.

The example below shows a group that contains two parameters identified by keywords:

```
[GROUP_NAME]
** This is a typical group definition.
KEYWORD1=VALUE1
KEYWORD2=VALUE2
```

In another example, the group [Server28] below illustrates a typical NetWeave communications configuration for a TCP/IP connection:

```
[Server28]
PROTOCOL=TCPIP
TCPIP_ADDRESS=localhost
TCPIP_PORT=myservice
```

You can specify configuration parameters for a given user function at the group level, at the keyword level, or in a few cases both. For example, the security parameters depend on settings in the [SECURITY] group. Data conversion parameters are defined by keywords in individual groups that describe files or messages. In general, when a parameter occurs in a group with a special name, you can often override it by making a change in another group that is specific to a file, message or application server.

Some parameters are important for defining the groups for each NetWeave library layer. Others are best understood in the context of the functions provided by a specific layer. This section describes parameters and syntax conventions that apply to groups in all layers of the library. The layer-specific parameters are discussed in the appropriate sections.

## The INI File Groups

### The [LICENSE\_GROUP] Group

The [LICENSE\_GROUP] has one entry, LICENSE\_KEY. A license key is required for each system that uses a NetWeave agent or on which a NetWeave server application runs. A NetWeave server is an application that calls `nwds_ipc_publish`, the function that listens for new TCP/IP connections. To receive TCP/IP connections from remote clients, an application must call `nwds_ipc_publish` with a group name in the configuration file that specifies either the IP address or the host name of the local system.

The LICENSE\_KEY is supplied by NetWeave technical support and encodes the host name or IP address and the expiration date of the current license. The license key is a string of uppercase alphabetic consonants.

There is one optional parameter for Tandem's TCP/IP, USE\_HOST\_FILE. Tandem has two functions to obtain the name of the local system: `gethostbyname` and `host_file_gethostbyname`. The former attempts to obtain the local host from a domain server. If none exists, the function eventually times out and looks up the host in `$system.system.hosts`, the Tandem equivalent of `/etc/hosts`. Unfortunately the timeout is long. The latter function looks in the hosts file first. Change the default search order by setting the optional parameter.

```
[LICENSE_GROUP]
LICENSE_KEY = QKFTRJJ
USE_HOST_FILE = 1
```

### The [TRACES] Group

The [TRACES] group controls how messages are generated and printed to an external log file. If your INI file does not include a [TRACES] group, NetWeave will not generate any messages or track errors.

Because a single external failure may affect more than one layer of the NetWeave library, each affected module may generate one or more error messages as it responds to error conditions reported from lower layers of the library. Whenever possible, NetWeave collects messages generated from a single failure and reports them in a single, multi-lined entry in the log.

We recommend that you configure your application to report errors. Because more than one application often uses the same trace file, you will want to report any application-specific errors in a separate log. When several applications share a log file, a single external problem at a remote host can cause errors to be reported simultaneously from all applications, making it hard for you to understand what happened to each application.

To configure your [TRACES] group parameters, you need to understand concepts such as *substitution strings*, the *boot INI file*, and the *root* or *start* group. Although this information is discussed in more detail elsewhere in this manual, this is a good place to introduce some of these other features and show how they work together.

In the example below, let's assume that two applications use the same INI file. Within the file, each application has a distinct root group – let's call them [APP1] and [APP2] – that declares the log file where the application's errors will be logged. The INI file would look like this:

```
[APP1]
```



```
@TRACE_FILE@=/usr/nwds/app1.log
[APP2]
@TRACE_FILE@=/usr/nwds/app2.log
*
*** Every boot INI file should have a [TRACES] group
*
[TRACES]
TRACE_LEVEL=ERRORS
FILE_NAME=@TRACE_FILE@
```

The table on the next page describes the most commonly used [TRACES] group parameters. For more information about the TRACE\_ARRAY parameter, see page 11.



Keyword	Value	Description
TRACE_LEVEL	ERRORS	Captures only NetWeave error information, but will consolidate all error reports from a common source.
	FULL	Prints all messages that the NetWeave modules generate. <b>NOTE:</b> Because this setting can quickly generate very large log files, use FULL only when you need to create a log file to send to NetWeave technical support for analysis.
	INFO	Reports all error and informational messages as they occur. Use INFO to prevent NetWeave from consolidating related error messages into a single entry in the log file.
	CLASS	Activates the TRACE_ARRAY feature that collects detailed trace information for selected modules.
TRACE_ARRAY	See the table below.	When advised by NetWeave technical support, use this parameter to capture all traces from selected layers (or code modules) in the NetWeave library.
FILE_NAME	A substitution name such as @TRACE_FILE@	Specifies the output device to which NetWeave logs the messages. It's a good idea to use a substitution name such as @TRACE_FILE@. You can redefine this name in the root group to allow multiple applications to share the same INI file, while each application still has a unique trace file.
TRACE_APPEND	0	Trace files are overwritten. This is the default value.
	1	New data is appended to the trace file.

During application testing, you may need to know more about the conditions that exist just before a reproducible error occurs. The TRACE\_SAVE feature directs NetWeave to capture program context in a sliding window whose size is the product of SAVE\_LINE\_SIZE and SAVE\_LINE\_COUNT. When an error occurs, NetWeave reports the contents of this window of detailed trace messages.

**CAUTION:** Because there is considerable additional overhead associated with TRACE\_SAVE, avoid using it during production.





Keyword	Default	Description
TRACE_SAVE	0	Setting TRACE_SAVE=1 modifies TRACE_LEVEL=ERRORS to collect and report additional detailed NetWeave traces that precede an error condition. Use this setting when the error traces don't give enough information to let you identify the context in which an error occurred.
SAVE_LINE_SIZE	256	Optional. Defines the maximum length (in bytes) of each trace message.
SAVE_LINE_COUNT	100	Optional. Defines how many lines to save in the sliding window.

NetWeave technical support can use `TRACE_ARRAY` to gather detailed trace information about a particular code module within the NetWeave library and then use this information to evaluate the problem. Use `TRACE_ARRAY` only when advised to do so.

TCLASS Value	Modules
20	L2_tcp.c, l2_tcpa.c, l2_tcpip.c
22	L2_dual.c
26	L2_bcast.c
27	L1_udp.c, l1_udpws.c
28	L1_udpnt.c
29	Callback.c
30	Level3.c
40	Ni_conne.c
50	Server.c
51	Rpcserve.c
55	Tansql.c
60	Ux_kern.c



## Message Log Facility

Programmers use NetWeave's message logging to integrate application messages with NetWeave trace messages. The messages in the log will be formatted according to the conventions of the `printf` function in C. For more information, see "nwds\_msglog" in the *NetWeave API Reference Guide*, and see "Application Message Logging" in this guide.

Message logging is an extension of the tracing facilities. Specify the configuration parameters for message logging in the TRACES group. The configuration settings determine which of the application's messages are printed in the log. In the table below, configuration settings are matched with the severity parameter in the `nwds_msglog` function. The TRACE level prints all messages. INFO prints all except those with severity level NWDS\_MLSTRACE. Other levels print successively fewer messages.

Keyword	Default	Description
MSGLOG_LEVEL	INFO	TRACE -- NWDS_MLSTRACE INFO -- NWDS_MLSINFO WARNING -- NWDS_MLSWARNING ERROR -- NWDS_MLSEERROR FATAL -- NWDS_MLSFATAL
MSGLOG_ID	none	This option applies only to NetWeave on IBM/CICS.

## The Root Group

The *root group*, sometimes called the *start group*, tells NetWeave where to start looking for the configuration information for a particular application. Because more than one application or Agent may use the same INI file, the root group is a convenient place to store private, application-specific information. An application declares the root group and INI file in the `nwds_init` call, and the root group is therefore the first group that NetWeave searches to resolve configuration issues. As noted in the previous section, every start group should contain a *substitution string* (such as `@TRACE_FILE@`) to designate a unique file to which NetWeave logs errors.

If you use the chaining method to link INI files in a hierarchy, the last entry in the root group must be the name of the next INI file in the hierarchy that you want NetWeave to use. In the example below, the start group `[APP1]` lists both a trace file substitution string and a link in a hierarchy of INI files. The next file in this chain of INI files is `common.ini`.

```
[APP1]
@TRACE_FILE@=/usr/nwds/app1.err
{/usr/nwds/common.ini}
```

The root group in a NetWeave Agent's INI file must contain a `PUBLIC_NAME` entry to specify the list of names by which the Agent is known to remote clients. An Agent hosted on a server that has more than one LAN connection should specify a public name for each LAN to which the server is attached. For example, the group `[NW_SERVER]` lists two groups (`[NT_AGENT1]` and `[NT_AGENT2]`) by which this Agent will be known to other applications.

```
[NW_SERVER]
PUBLIC_NAME={NT_AGENT1, NT_AGENT2}
@TRACE_FILE@=server.err
[NT_AGENT1]
PROTOCOL      =TCPIP
TCPIP_ADDRESS=SERVER1
TCPIP_PORT    =18476
[NT_AGENT2]
PROTOCOL      =TCPIP
TCPIP_ADDRESS=SERVER2
TCPIP_PORT    =18476
```



## Naming Services

You can use any of the following NetWeave features to isolate programs from details of the physical environment in which they run:

- Use a logical name (an *alias*) in the NetWeave API parameters that expect a file name or process name.
- *Define a route* that an application message uses to reach either the application or the Agent that services it.
- Define a logical name to refer to a collection of alternate routes to an Agent, a file, or a server. This feature is referred to as *load balancing*.
- Use the NetWeave API and INI files to store and access the configuration information that an application needs.
- Use *substitution strings* to simplify deployment. These strings provide a placeholder or variable name in one definition that can be specified in different ways by applications that share the INI file.

### Logical Names and Aliases

To enhance the flexibility of the NetWeave API, you can use either a file name or process name alias as a parameter in a NetWeave call. For example, a call to `nwds_file_open` may accept either the alias (which NetWeave will translate) or the physical file name. Use the actual physical file name only in the INI file of the Agent or application server that uses the file directly. Use an alias for the file in the remote client's INI file.

In the example below, the group name `[MyFile]` refers to a physical file named `/usr/mydata/file1.dat`:

```
[MyFile]
NAME=/usr/mydata/file1.dat
```

The program code references `MyFile`, and at runtime NetWeave determines from the INI file which file `MyFile` refers to. The special keyword `NAME` has several uses in NetWeave configurations. In this example, it creates an alias (`MyFile`) for the physical filename.

### Message Routing

The special keyword `NAME` is also used to define the logical route that a message travels to reach the server that processes it. A route is a series of links from one NetWeave process or Agent to the next. The last link in the route connects to the target for the service request. To declare a single link in a route, NetWeave uses a special syntax (a double colon) to specify a target service at the next node in the route. The example below defines a route to `<TARGET>` via `<NODE>`, with `<NODE>` standing for an arbitrary node definition, and `<TARGET>` referring to a target service:

```
NAME=<NODE>::<TARGET>
```

Because a NetWeave Agent performs all operations on NetWeave FIFO queues, configuration references to FIFOs frequently use this syntax. In the example below, the group defines a FIFO named Q1 that is mediated by a NetWeave Agent named Sun3. The target Q1 is another group in the Agent's INI file that is also named Q1. Of course you can name it anything you like, but it is easier to track a route through several hops if the target takes the same name as the group that declares the hop. The application's INI file contains the following:

```
[Q1]
NAME=Sun3::Q1
```

And the Agent named Sun3 might have an alias defined for Q1:

```
[Q1]
NAME=/usr/netweave/fifo/q1.dat
```

Or, it could declare another hop to a second Agent:

```
[Q1]
NAME=Alpha2::Q1
```

## Load Balancing

Load balancing distributes connections evenly across a set of servers. The servers in the specified group may be any valid connection group that NetWeave supports: they may be on the same machine or on different machines, across different protocols, or have dual-rail connections.

In the example below, [Service-A] consists of two servers, S1 and S2. When a client application connects to [Service-A], the NetWeave library randomly chooses one member from the set of servers and connects to it. (NetWeave does not use a round-robin selection.) Let's assume that one or more client applications share the same configuration of [Service-A], and that together these clients connect to [Service-A] 100 times. Approximately 50 of the client connections would attach to S1, and 50 would connect to S2. The maximum length of the NAME line limits the maximum number of servers in a service.

```
[SERVICE-A]
NAME={S1, S2}
[S1]
PROTOCOL=.....

[S2]
PROTOCOL=.....
```

If servers require different proportionate loading, entries in the list may be repeated to create the desired ratio. The declaration in the example below assigns 40% of the load to S1 and 60% to S2. To continue the example from above, the clients would connect to S1 about 40 times and to S2 about 60 times.

```
[SERVICE-A]
NAME={S1, S1, S2, S2, S2}
```

You can also use the load balancing feature to instruct NetWeave to retry another connection when one fails. When the `NAME` parameter has a single value, NetWeave does not retry a connection. However, if `NAME` has multiple values and a selected connection fails, the load balancing feature ensures that NetWeave will choose another connection and try again. The parameter `LB_RETRIES` in the well-known group `RPC_CLIENT` controls how many times NetWeave will retry a failed connection. The default is 5.

Even if you have only one server or Agent to supply your service, you can use proportionate loading to declare load balancing, and to declare that you want connections retried automatically. This can be helpful if the connection failure is a temporary condition caused by an internal network timeout. If the cause is not transient, repeated retries only delay an inevitable report of failure. Although the declaration below may look a little odd, it tells NetWeave to use proportionate loading and to retry a connection to `S1`:

```
[SERVICE-A]
NAME={ S1, S1 }
```

## Load Balancing Example

Assume there is a service called `XYZ`. Because it requires high parallel processing capacity, this service must be implemented using three physically separate but functionally equivalent server processes that may run on different physical computers:

- Two processes (`XYZ1` and `XYZ2`) running on one server machine
- One process (`XYZ3`) running on a second server machine

These server machines can be accessed through DNS using the names `SERVER1` and `SERVER2`, respectively, and require the following three INI files (fragments only):

### SERVER1 machine

```
[XYZ1]
PROTOCOL=TCPIP
TCPIP_ADDRESS=SERVER1
TCPIP_PORT=12345
```

```
[XYZ2]
PROTOCOL=TCPIP
TCPIP_ADDRESS=SERVER1
TCPIP_PORT=23456
```

etc...

### SERVER2 Machine

```
[XYZ3]
PROTOCOL=TCPIP
TCPIP_ADDRESS=SERVER2
TCPIP_PORT=34567
```

etc...

### Client Machines

```
[XYZ]
NAME={XYZ1, XYZ2, XYZ3}
```

```
[XYZ1]
PROTOCOL=TCPIP
TCPIP_ADDRESS=SERVER1
TCPIP_PORT=12345
```

```
[XYZ2]
PROTOCOL=TCPIP
TCPIP_ADDRESS=SERVER1
TCPIP_PORT=23456
```

```
[XYZ3]
PROTOCOL=TCPIP
TCPIP_ADDRESS=SERVER2
TCPIP_PORT=34567
```

etc...

## Configuration Information for Applications

For storing and retrieving keyword values, use the following NetWeave functions:

- `nwds_ini_get_name`
- `nwds_ini_put_name`
- `nwds_ini_delete_name`
- `nwds_ini_get_int`

Conceptually these functions operate as simple database calls. For example, `nwds_ini_put_name` stores a value by the keyword (the name). For a detailed description of these functions, please see the *NetWeave API Guide*.

To have your application retrieve run-time information from the same file that NetWeave uses, use the NetWeave function `nwds_ini_get_name`. This allows all configuration – both NetWeave- and application-specific parameters – to be declared in one place. And you can use NetWeave's dynamic INI files to modify application-specific parameters while the program is running.

The NetWeave special group `[USER_NAME_GROUP]` contains application-specific data. Use this group to store information for your applications. For example, a call to `nwds_ini_get_int` (“TheBeast”) might return 666 to an application.

```
[USER_NAME_GROUP]
TheBeast=666
```

NetWeave also uses `USER_NAME_GROUP` as a special keyword to identify a configuration group that contains parameters specific to a particular application. For example, the group `[Adam]` has `[Eve]` as its `USER_NAME_GROUP`. An application that calls `nwds_init` with start group `[Adam]` will return 17 when it calls `nwds_ini_get_int` (“TheBeast”).

```
[Adam]
USER_NAME_GROUP=Eve
[Eve]
TheBeast=17
```

## Substitution Strings

To simplify the deployment of INI files from one platform to another, you can create macro definitions, also referred to as substitution strings, which replace one character string with another. Use the character `@` to delimit a substitution string as shown below:

- `@String2@=String4`
- `Keyword=String1 @String2@String3`  
This parameter’s value is `String1` followed by `String4` followed by `String3`, i.e., `keyword=String1String4String3`.
- Substitution strings are most often used to define a unique trace file for each application. In the application’s start group you define a macro (by convention, it is `@TRACE_FILE@`) that is referenced in the `[TRACES]` group:

```
[NW_SERVER]
PUBLIC_NAME={NT_AGENT}
@TRACE_FILE@=server.err
[TRACES]
TRACE_LEVEL=ERRORS
FILE_NAME=@TRACE_FILE@
```



## Configuring Sessions and Services

This section describes how NetWeave handles data conversion and compression.

### Data Conversion

The settings in the INI file determine how and when NetWeave translates records or messages. For example, if the INI file settings enable translation for a remote file group, NetWeave translates these records automatically. To translate a message, NetWeave needs both the INI file and item list entry information. The sender or the receiver may control message translation using the item type `NWDS_IPC_CONVERT_NAME` in either `nwds_ipc_read` or `nwds_ipc_write`.

NetWeave can translate the following data types:

Data type	Description
SHORT	System- and compiler-dependent range (a 16-bit integer).
LONG	System- and compiler-dependent range (a 32-bit integer).
CHAR	A fixed-length binary stream.
STRING	A NULL-terminated array of (usually printable) characters.

Three keywords control translation within a group:

Keyword	Description
DDL_ENTRY	To enable translation, set <code>DDL_ENTRY=1</code> .
DDL_FIELD_COUNT	The number of fields in the record or message.
DDL_FIELD_nn	The data type of the field indexed by nn.

A typical example of data conversion settings:

```
DDL_ENTRY=1
DDL_FIELD_COUNT=3
DDL_FIELD_1="LONG 4"
DDL_FIELD_2="SHORT 2"
DDL_FIELD_3="CHAR 4"
```



## Data Compression

NetWeave's data compression option can improve transmission speeds in wide-area networks that have low underlying bandwidth. When a slow modem limits message throughput (in X.25 networks and asynchronous dial-up connections, for example), decreasing the amount of data being transmitted also decreases the total transmission time and generally improves the perceived response time.

**NOTE:** The compression option is not recommended for short messages sent through local area networks. The time it takes the sending and receiving computers to compress and decompress the data exceeds any time saved by transmitting less data.

NetWeave uses two common data compression algorithms, the Huffman and the Ziv-Lempel (LZ). As you set your compression and decompression values, keep in mind the following:

1. The Huffman algorithm provides much faster compression.
2. The Ziv-Lempel algorithm provides a higher compression ratio as well as somewhat faster decompression.
3. In point-to-point connections for messages of modest length, the Huffman algorithm's faster compression time compensates for its poorer compression ratio.
4. In multicast situations, Ziv-Lempel's compression ratios and decompression times are superior to Huffman's.
5. Compression and decompression times depend heavily on processor load. Do not expect the maximum speed in a production environment.
6. Do not use compression to save a few milliseconds of transmission time. The cost in processor load outweighs any savings in transmission.

## Data Compression Parameters

Compression is an all or nothing proposition. Because the compression parameters are described in the group [LEVEL4], compression applies either to all messages sent to and from applications that use an INI file, or it applies to none. For this reason, NetWeave recommends you enable compression only through definitions in the boot INI files, and make sure that both sender and receiver use the same settings.

Keyword	Range	Default
COMPRESSION_ON	0=off nonzero=on	0
HUFFMAN_LOWER	1 - 32000	500
HUFFMAN_UPPER	1 - 32000	5000
LZ_LOWER	1 - 32000	5001
LZ_UPPER	1 - 32000	30000

## Data Compression Examples

Each compression algorithm accepts a lower and an upper limit to define the range of message length (in bytes) that will be compressed. Messages that are too long or too short will not be compressed. The choice of compression algorithm depends on which algorithm's bounds can better accommodate the message size. If both algorithms apply, NetWeave uses the one that yields the shorter message.

The example below shows how to set the parameters for:

- No compression of messages shorter than 1K
- Huffman encoding for messages less than 4K bytes
- Ziv-Lempel encoding for all others

```
[LEVEL4]
COMPRESSION_ON=1
HUFFMAN_LOWER=1000
HUFFMAN_UPPER=3999
LZ_LOWER=4000
LZ_UPPER=32000
```

If you don't want to use a particular algorithm, set its upper bound value to 1. The example below shows how to use Ziv-Lempel to compress all messages larger than 2 KB. (Huffman encoding is disabled.)

```
[LEVEL4]
COMPRESSION_ON=1
HUFFMAN_LOWER=1000
HUFFMAN_UPPER=1
LZ_LOWER=2000
LZ_UPPER=32000
```

## Algorithm Technical Information

The two algorithms that NetWeave uses are based on information in *The Data Compression Book* by Mark Nelson (M&T Books, 1992) and both are in the public domain.

One algorithm is a version of the Huffman algorithm that provides minimum redundancy coding. It is a static compression algorithm: phrases are replaced by tokens, and when the number of bits in the phrase exceeds the number of bits in the token, NetWeave compresses the data. The Huffman algorithm creates variable-length codes that contain an integral number of bits.

The other algorithm uses the Ziv and Lempel sliding window technique to implement an adaptive encoding algorithm. An adaptive algorithm adjusts to give maximum compression to the phrases that occur most often. To do this, information about previous portions of a message is stored in a temporary "dictionary" that contains a set of fixed-length phrases found in a window into the previously processed text. The window size can range from 2K - 16K, and the length of the phrases from one or two to as many as 16 bytes.

The tables below list the results of compression tests of sample data. These tests were performed on a Sun SPARCstation 10 running the Solaris operating system. If you use other hardware your results will vary somewhat, but the relationship among the values should parallel what is shown below. Please note:

- Each line in the table represents an average value calculated for four files of a particular size. Two of the files are C language source code; the other two are binary files.
- The compression ratio is  $(1 - (\text{compressed\_size} / \text{raw\_size})) * 100 \%$ .
- The compression and decompression times show how long it took to execute the actual compression and decompression phases. The total time shown in the last column is the sum of the compression and decompression times in the two previous columns.

## Huffman

File size (Kbytes)	Compression ratio (%)	Compression time (seconds)	Decompression time (seconds)	Total time (seconds)
0.5	23.0	0.004	0.005	0.009
1.0	27.6	0.006	0.007	0.013
1.5	30.7	0.007	0.009	0.016
2.0	32.3	0.008	0.011	0.019
2.5	31.1	0.009	0.013	0.022
3.0	33.7	0.010	0.014	0.024
5.0	34.7	0.015	0.021	0.036
10.0	36.7	0.025	0.038	0.063
15.0	37.8	0.034	0.055	0.089
20.0	38.5	0.044	0.072	0.116
25.0	39.0	0.052	0.086	0.138
30.0	39.5	0.061	0.108	0.169



## Ziv-Lempel

<b>File size (Kbytes)</b>	<b>Compression ratio (%)</b>	<b>Compression time (seconds)</b>	<b>Decompression time (seconds)</b>	<b>Total time (seconds)</b>
0.5	35.9	0.008	0.001	0.009
1.0	36.2	0.019	0.002	0.021
1.5	39.1	0.027	0.003	0.030
2.0	41.6	0.036	0.004	0.040
2.5	44.7	0.048	0.005	0.053
3.0	46.2	0.060	0.006	0.066
5.0	52.2	0.110	0.009	0.119
10.0	56.4	0.200	0.016	0.216
15.0	58.6	0.305	0.025	0.330
20.0	60.3	0.410	0.028	0.438
25.0	60.8	0.530	0.036	0.566
30.0	61.8	0.610	0.044	0.654



## NetWeave Security Services

NetWeave security functions include *authentication*, *access control*, and *encryption*. Authentication is the process of verifying that a client is who it claims to be. NetWeave supports two distinct authentication mechanisms:

- Authentication by name and password attempts to validate the client using host operating system calls to the host's security tables. This type of authentication requires the services of a NetWeave Agent on the host.
- Challenge-response authentication is more secure than authentication by name and password, and does not require the presence of an Agent to mediate the authentication process. For challenge-response, the NetWeave libraries in the server and client conduct a secret negotiation (transparent to the client and server applications) to prove the client's identity.

The NetWeave access control procedure determines whether a remote client has access rights to open a host file. This procedure uses the host's operating system calls to check the client's access rights. A NetWeave Agent on the host mediates these rights.

Encryption protects a message from being read by anyone other than its intended readers. NetWeave provides several encryption algorithms for encoding and decoding messages. All encryption algorithms use private keys.

**NOTE:** The password obfuscation feature, which operates independently of the Encryption feature, disguises the password passed in the `nwds_password` API. In older versions of NetWeave, obfuscation was optional and controlled by an INI file parameter in the Security Group. It is now automatic and passwords are always disguised.

### Authentication By Name and Password

The NetWeave API for connecting a remote client to a host computer is an interface with the standard terminal validation functions of the host's operating system. Authentication by username and password is intended to simulate logging in through a remote terminal. The NetWeave Agent has the central role in authentication by username and password because only the Agent can access the appropriate tables on the host. A remote client enters a username and password that the Agent checks against the host system's security tables.

The following API calls are used for authentication by username and password:

Function	Description
<code>nwds_logon</code>	Presents the user's name to the NetWeave Agent.
<code>nwds_password</code>	Presents the user's password to the Agent. If the Agent accepts the logon and password as valid, it starts a session for the user.
<code>nwds_logoff</code>	Declares the user's intention to end the current session.

The [SECURITY] group in the NetWeave Agent's INI file controls username and password authentication:

```
[SECURITY]
SECURITY_ON=1
```

The default is no security (SECURITY\_ON=0). If the [SECURITY] group is not defined, or if SECURITY\_ON is missing (or commented out), then the Agent runs without security.

## Authentication Details By Platform

### Windows OS

On the PC, NetWeave manages authentication using the [SECURITY] group entries in the Agent's INI file. The [SECURITY] group contains a list of user IDs and passwords in the following format:

```
user id=password
```

After authentication by the Agent, a user has full access to all files and resources to which the Agent process has access. When a file is opened on a client's behalf, there is no additional check to determine whether the client has access rights to that file. A sample PC security entry for user JOHNDOE (using the password "secret") is shown below:

```
[SECURITY]
SECURITY_ON=1
JOHNDOE=secret
```

### UNIX

The NetWeave Agent uses the UNIX account database to verify the user name and password. To process user requests, the Agent performs a `setuid()` system call to assume the user's ID – and thus is given only the access capabilities that the client has. NetWeave does not verify file access capabilities independently. Instead it lets the operating system determine access rights when the Agent, in the guise of the client, tries to open the file.

### Tandem Guardian OS

On Tandem, NetWeave uses the system call `USER_AUTHENTICATE_` to verify a client's authenticity. To determine whether a client may access legacy (Enscribe) and FIFO files on Tandem Guardian system, NetWeave uses the system call `USERNAMETOUSERID`. For example, an application on a remote workstation may open and access an Enscribe file only if it logs on as a user who has access rights consistent with Guardian conventions. These security features are optional.

**NOTE:** NetWeave's use of Guardian functions to control access to files does not apply to C files or to NonStopSQL tables on Tandem.

## Digital VAX/VMS and OpenVMS

To verify a client's authenticity, NetWeave uses the system call `sys$getuai`. To control access to legacy (RMS) files on VMS and OpenVMS, NetWeave uses the system call `sys$check_access`.

## IBM/CICS

The NetWeave Agent is not implemented for this release of IBM/CICS. Because the Agent controls client verification and file access rights, these functions are not implemented for IBM/CICS.

## Authentication By Challenge-Response

Challenge-response authentication offers better protection than authentication by name and password. For challenge-response, the server and client conduct a secret negotiation (transparent to the client and server applications) to prove the client's identity. Challenge-response authentication does not require any coding changes, and it does not require a NetWeave Agent on the host.

**NOTE:** In release 2.0, challenge-response authentication requires encryption. If you want to use challenge-response, be aware that all messages between the client and server will be encrypted, not just the secret negotiations required to authenticate the client. This dependency will be removed in a future release of NetWeave.

Here's how challenge-response authentication works:

1. The client sends its name to the server (or NetWeave Agent).
2. The server looks up the client's name in its INI file and retrieves the client's private key.
3. The server generates a random message and encodes it with the client's key.
4. The server sends this encoded message (the challenge) to the client.
5. The client decodes the challenge and returns the decoded message as its response.
6. If the decoded message matches the original random message, access is granted.

To configure challenge-response for a client, add the following parameters to the *server connection group* in the client's INI file.

```
CHALLENGE_RESPONSE=1
ENCRYPTION_REQUIRED=1
ENCRYPTION_TYPE=NWE1
LOGON_NAME=PS0
PS0=abcdefghijkl
```

The `LOGON_NAME` tells the server who the client is. To find the client's private key, NetWeave uses the value of the `LOGON_NAME` parameter as a keyword. In this example, the `PS0` parameter is the client's private key, which should be 12 characters in length. If it is shorter, NetWeave automatically pads it to 12. If it is longer, the system ignores the extra characters.

**NOTE:** A key of 12 characters is equivalent to encoding with a key length of 84 bits.



To configure challenge-response for the server, add the following parameters to the *public* (also called *publish*) *group* in the server's INI file:

```
CHALLENGE_RESPONSE=1
ENCRYPTION_REQUIRED=1
ENCRYPTION_TYPE=NWE1
```

In addition to these required parameters, you must add a parameter for each client that will be permitted to connect to this server. These additional parameters look something like this:

```
PS0=abcdefghijkl
PS1=bcdefghijklm
PS2=cdefghijklmn
PS3=defghijklmno
PS4=efghijklmnop
PS5=fghijklmnopq
PS6=ghijklmnopqr
PS7=hijklmnopqrs
PS8=ijklmnopqrst
PS9=jklmnopqrstu
```

**NOTE:** In release 2.0, clients' private keys are stored in INI files in the clear. In the next release, these keys will be stored in encoded form in a special file.

## Encryption

The table below lists the three NetWeave encryption types:

Encryption type	Description	Other required parameters
NWE1	The standard public domain algorithm, provided at no additional charge.	None
DES	XyGate/SE DES algorithm for 56-bit keys. There is an additional license charge for this algorithm.	DIFFIE_HELLMAN_KEYSIZE
3DES	XyGate/SE Triple DES algorithm for 168-bit keys. There is an additional license charge for this algorithm.	DIFFIE_HELLMAN_KEYSIZE

**NOTE:** As part of an Embedder's Agreement with Xypro Technology Corporation in California, Vertex bundles the XyGate/SE Encryption component with its NetWeave middleware to provide for DES and Triple DES (3DES) encryption schemes. The XyGate/SE package is currently supported on PCs running Windows or NT, UNIX, and Tandem.

Individual sessions may use different encryption algorithms. To enable encryption, add the following line to the logical group that defines the connection parameters:

```
ENCRYPTION_REQUIRED=1
```

In the example below, the connection group TEST1 uses DES encryption and a Diffie-Hellman key of 512 bits:

```
[TEST1]
PROTOCOL=TCPIP
TCPIP_ADDRESS=198.198.7.122
TCPIP_PORT=17895
ENCRYPTION_REQUIRED=1
ENCRYPTION_TYPE=DES
DIFFIE_HELLMAN_KEYSIZE=512
```

Please note the following:

- Both the client and the application to which it connects must have identical information in their INI files.
- Encryption may be applied only to sessions between client and server applications, or between a remote client and a NetWeave Agent. Do not apply encryption to sessions between applications located on the same system.
- If you are using DES or 3DES, the XyGate key file and the executable must be located in the same folder.

## Parameters for XyGate Encryption Algorithms

Both DES and TripleDES (3DES) accept the `DIFFIE_HELLMAN_KEYSIZE` parameter, which affects the time required to initialize the keys for a given session. The longer the key, the longer it takes to connect to the peer application. The default length is 1024 bits.

## XyGate Status Codes

The XyGate encryption library is embedded within the NetWeave library. Errors from XyGate are returned to NetWeave and reported in the NetWeave error log as numeric values that can be interpreted as follows:

```
#define CRYPT_OK                0                /* No error */

/* Internal errors */

#define CRYPT_ERROR             -1                /* Nonspecific error */
#define CRYPT_SELFTEST         -2                /* Failed self-test */

/* Error in parameters passed to function */
```



```
#define CRYPT_BADPARAM          -10      /*Generic bad argument to function */
#define CRYPT_BADPARAM1        -11      /*Bad argument, parameter 1*/
#define CRYPT_BADPARAM2        -12      /*Bad argument, parameter 2*/
#define CRYPT_BADPARAM3        -13      /*Bad argument, parameter 3*/
#define CRYPT_BADPARAM4        -14      /*Bad argument, parameter 4*/
#define CRYPT_BADPARAM5        -15      /*Bad argument, parameter 5*/
#define CRYPT_BADPARAM6        -16      /*Bad argument, parameter 6*/
#define CRYPT_BADPARAM7        -17      /*Bad argument, parameter 7*/
#define CRYPT_BADPARAM8        -18      /*Bad argument, parameter 8*/
#define CRYPT_BADPARAM9        -19      /*Bad argument, parameter 9*/
#define CRYPT_BADPARAM10       -20      /*Bad argument, parameter 10*/
#define CRYPT_BADPARAM11       -21      /*Bad argument, parameter 11*/
#define CRYPT_BADPARAM12       -22      /*Bad argument, parameter 12*/
#define CRYPT_BADPARAM13       -23      /*Bad argument, parameter 13*/
#define CRYPT_BADPARAM14       -24      /*Bad argument, parameter 14*/
#define CRYPT_BADPARAM15       -25      /*Bad argument, parameter 15*/

/* Errors due to insufficient resources */


#define CRYPT_NOMEM             -50      /* Out of memory*/
#define CRYPT_NOTINITED        -51      /* Data has not been initialized*/
#define CRYPT_INITED           -52      /* Data has already been
initialized*/
#define CRYPT_NOALGO           -53      /* Algorithm unavailable*/
#define CRYPT_NOMODE           -54      /* Encryption mode unavailable*/
#define CRYPT_NOKEY            -55      /* Key not initialised*/
#define CRYPT_NOIV             -56      /*IV not initialised*/
#define CRYPT_NOLOCK           -57      /*Unable to lock pages in memory*/
#define CRYPT_NORANDOM         -58      /*No reliable random data available*/
#define CRYPT_BUSY             -59      /*Context is busy (async operation)*/

/* CAPI security violations */

#define CRYPT_NOTAVAIL         -100     /*Operation not available for this
algo/mode*/
#define CRYPT_KEYPERM          -101     /*No key permissions for this operation*/
#define CRYPT_WRONGKEY         -102     /*Incorrect key used to decrypt data*/
#define CRYPT_INCOMPLETE       -103     /*Operation incomplete/still in progress*/
#define CRYPT_COMPLETE         -104     /*Operation complete/can't continue*/
#define CRYPT_ORPHAN           -105     /*Encryption contexts remained allocated*/

/* High-level function errors */

#define CRYPT_DATASIZE         -150     /*Too much data supplied to function*/
#define CRYPT_PKCCRYPT         -151     /*PKC en/decryption failed*/
#define CRYPT_BADDATA          -152     /*Bad data format in object*/
#define CRYPT_BADSIG           -153     /*Bad signature on data*/
```



```
/* Key collection errors */
```

```
#define CRYPT_KEYSET_OPEN          -200  /*Cannot open key set*/
#define CRYPT_KEYSET_NOTFOUND      -201  /*Key or key info not found in key set*/
#define CRYPT_KEYSET_DUPLICATE     -202  /*Key already present in key set*/
#define CRYPT_KEYSET_READ          -203  /*Cannot read data from key set*/
#define CRYPT_KEYSET_UPDATE        -204  /*Cannot update key set*/
```

```
/* Data enveloping errors */
```

```
#define CRYPT_ENVELOPE_OVERFLOW.  -250  /*Too much data in envelope*/
#define CRYPT_ENVELOPE_UNDERFLOW -251  /*Too little data in envelope*/
#define CRYPT_ENVELOPE_RESOURCE   -252  /*Need resource to proceed*/
```

```
/* Certificate errors */
```

```
#define CRYPT_CERT_NONE           -300  /*No errors defined yet*/
```



## The NetWeave Communications Stack

The previous sections explained how to use the parameters that configure the NetWeave functions of the session or services layer. This section describes the parameters of the communications stack, the layer between the session and the individual communications protocols. Most of the communications stack parameters directly affect the performance of an application.

In this section, the word *connection* refers to a physical link between two computers. Parameters at the protocol layer define how to configure connections. The word *session* refers to a logical link between applications on two computers. A connection may support more than one session at a time.

### Using Heartbeats to Monitor a Communications Stack

NetWeave uses heartbeats for timely detection and reporting of certain types of failures in the underlying network connection. When properly set, heartbeats improve an application's ability to detect and react to failures, and thereby improve the application's throughput and reliability. Because heartbeats are confined to the lower levels of NetWeave's communications stack, applications are never aware of them directly.

The heartbeat mechanism provides an efficient way to verify that a connection is still active. Let's assume that two NetWeave-enabled applications establish a session with each other. Each time a message is sent from one to the other, NetWeave starts the heartbeat timer. If the timer expires, it means the sending application has not sent another message since the timer was started. In order to regularly reset the timer, the NetWeave communications stack automatically sends a heartbeat message, a tiny message to show that this side of the connection is still active.

This example considered a heartbeat only from the message sender's point of view. However, you can also set a timeout that reflects the receiver's point of view. If you intend to use heartbeats, set both types of timers as follows:

- Use keyword `SEND_HEARTBEAT_TIMER` to specify the length (in seconds) to wait before the communications stack automatically sends a heartbeat message.
- Use keyword `RCV_HEARTBEAT_TIMER` to specify the length (in seconds) to wait for any message from the sender.

You can set the timer parameters either in the protocol group that defines a specific connection, or in the special well-known group `[LEVEL3]`. Although parameters set in the `[Level3]` group apply to all connections defined at the protocol layer, the heartbeat parameters at the protocol layer override the `[Level3]` settings. To use the heartbeat mechanism, you must set `HEARTBEAT_REQUIRED=1` at the protocol layer for each connection that is to be monitored.

**CAUTION:** Use the heartbeats feature only with *asynchronous* applications. Using heartbeats with synchronous programs will cause the partner applications and/or agents to terminate.

**CAUTION:** To provide reliable message delivery over UDP/IP, the *broadcast protocol* uses its own heartbeat mechanism between NetWeave Agents. If an Agent is configured for the NetWeave broadcast protocol, do not enable the generic heartbeat mechanism in the `[LEVEL3]` group.

**NOTE:** NetWeave releases earlier than 1.05.05 do not support heartbeat in the communications stack. Subsequent releases detect when they are communicating with an older release, and disable heartbeats on those connections.

Use the following parameters to set the heartbeats:

Keyword	Default	Description
HEARTBEAT_REQUIRED	0	Optional. To enable the heartbeat feature, set this parameter to any non-zero value. <b>NOTE:</b> This parameter affects only the protocol layer.
RECV_HEARTBEAT_TIMER	120	The number of seconds the receiving side waits to hear from a partner before it terminates the open connection(s). This value should be two to three times larger than SEND_HEARTBEAT_TIMER.
SEND_HEARTBEAT_TIMER	60	The number of seconds that the sending system waits after sending the last message before emitting a heartbeat message.

## Using Capping and Pooling to Control How Connections are Used

To conserve system resources, NetWeave attempts to minimize connections at the protocol layer and instead interleaves (multiplexes) several sessions over a single connection. A connection is initially established to accommodate the first session, and as other sessions are created, their traffic is added on. When the last of these sessions terminates, the connection terminates too.


*Capping* limits the number of sessions that may share a connection. If your sessions tend to have high volumes of message traffic, a single physical connection may not have enough bandwidth for several such sessions. By setting a cap on the number of simultaneous sessions, you avoid flooding the connection with too much traffic.

When a cap is in effect, if NetWeave reaches the limit of permissible sessions, every additional attempt to establish a session will fail. When this happens, NetWeave automatically creates (if possible) a new connection to handle the additional sessions.

**NOTE:** Capping is not a magic solution. If your network's underlying physical bandwidth is inadequate for the expected session rates, it won't help to cap the number of sessions and create more connections over the same physical LAN. Capping can help only when connections are logically related to a specific physical path or to a specific CPU. For example, capping is effective when two computers, each consisting of a cluster of CPUs, communicate over a network that can create multiple independent physical paths between them.

To set the cap limit, use the parameter `MAX_CIRCUITS`. You can define the capping parameters generically at the network layer by changing the settings in the `[Level3]` group, or you can cap individual connections at the protocol layer. If you set `MAX_CIRCUITS` in `[Level3]`, the cap applies to all connections, unless you override it by setting different caps for individual protocol groups.

In the sample configuration below, connection groups `[GROUP1]` and `[GROUP2]` are configured with `MAX_CIRCUITS` of 2.



```
[GROUP1]
PROTOCOL=TCPIP
TCPIP_ADDRESS=localhost
TCPIP_PORT =19130
```

```
[GROUP2]
PROTOCOL=TCPIP
TCPIP_ADDRESS=localhost
TCPIP_PORT =19130
```

```
[LEVEL3]
MAX_CIRCUITS=2
```

In the example below, because the sample configuration has no `MAX_CIRCUITS` definition in `[Level3]`, `[GROUP1]` is configured with two circuits per physical link and `[GROUP2]` is configured with three. If `[Level3]` did contain a `MAX_CIRCUITS` definition, this configuration would apply to all connection groups, unless you specifically overrode it by setting group-specific values within a connection group.

```
[GROUP1]
PROTOCOL=TCPIP
TCPIP_ADDRESS=localhost
TCPIP_PORT=19130
MAX_CIRCUITS=2
```

```
[GROUP2]
PROTOCOL=TCPIP
TCPIP_ADDRESS=localhost
TCPIP_PORT=19130
MAX_CIRCUITS=3
```

```
[LEVEL3]
```


```
...
```

*Pooling* extends the lifetime of a connection by keeping it active even after the last session on it ends. If many sessions intermittently use a common connection, there may be brief intervals when no session is using the connection. Because it is more expensive to destroy and then recreate a connection than it is to leave it in place for a short period of time, pooling provides a mechanism for using a connection more efficiently.

To activate the pooling feature, use the setting `LINK_POOLING=1`. Use `LINK_POOL_TIMER` to specify how long (in seconds) the connection should stay active. The default is 300 seconds.

You can define the pooling parameters in the `[Level3]` group, or you can set pooling for individual connections at the protocol layer. If you set pooling in `[Level3]`, it applies to all connections.

In the sample configuration below, connection groups `[GROUP1]` and `[GROUP2]` are configured with `LINK_POOLING` set on, and a timeout value of 60 seconds.



```
[GROUP1]
PROTOCOL=TCPIP
TCPIP_ADDRESS=localhost
TCPIP_PORT =19130
```

```
[GROUP2]
PROTOCOL=TCPIP
TCPIP_ADDRESS=localhost
TCPIP_PORT=19129
```

```
[LEVEL3]
LINK_POOLING=1
LINK_TIMEOUT=60
```

In the example below, connection group [GROUP1] is enabled with link pooling and a timeout of 60 seconds. Connection group [GROUP2] is not configured with link pooling. If you specify parameters for LINK\_POOLING in the [Level3] group, they will apply to [GROUP2] but not [GROUP1].

```
[GROUP1]
PROTOCOL=TCPIP
TCPIP_ADDRESS=localhost
TCPIP_PORT=19130
LINK_POOLING=1
LINK_TIMEOUT=60
```

```
[GROUP2]
PROTOCOL=TCPIP
TCPIP_ADDRESS=localhost
TCPIP_PORT=19129
```

```
[LEVEL3]
```

```
...
```

## Connection Timeouts

The parameter `CONNECT_TIMEOUT` may be specified in either the `LEVEL3` group or in a protocol group to limit how long NetWeave will wait for a non-blocking call to `nwds_ipc_connect` to complete. When declared in `LEVEL3`, the timeout value (in seconds) applies to all protocols. When specified in a protocol group, it overrides the value, if any, declared in the `LEVEL3` group. Connection timeouts do not apply to blocking, synchronous connections.



## The Protocol Layer

The protocol layer is the lowest layer in the NetWeave architecture. The protocol layer is important because this is where NetWeave interacts with communications resources in the computer system. To form a connection between a client application on one platform and a server on another, the INI files on each must have identical entries for the connection group they will use to make the connection. You can set the protocol parameters either generically in the [LEVEL2] group, or individually within each connection group. (A connection group is also referred to as a protocol group.)

### TCP/IP Protocol

All NetWeave platforms support the TCP/IP protocol, which is used to make point-to-point connections between pairs of IP addresses and ports. Each computer in the network has at least one IP address, and each address has many ports associated with it. If a computer is attached to more than one physically distinct LAN, it has more than one IP address.

You can specify IP addresses and ports either as logical names, or as numerical addresses. Logical address names are stored in the system's HOST file, while logical port names are stored in the SERVICES file. For a numerical address, use the standard IP form *dotted octets*. For example, 127.0.0.1 is the IP address of the computer known as localhost.

For some computer systems, such as Tandem or IBM/CICS, you must specify a few additional TCP/IP parameters. For example, Tandem requires a TCPIP\_PROCESS\_NAME that identifies the TCP/IP process associated with a particular IP address. For more information about the TCP/IP parameters for IBM/CICS, see the *NetWeave Installation and Operations Guide for IBM/CICS*.

Some parameters for TCP/IP are stored in the group [LEVEL2] and apply to all TCP/IP connections that are defined in the INI file. Parameters for a specific connection are specified in an individual connection group. Not all parameters are used on all platforms. In the table below, LEVEL2=Yes means that the parameter, if specified, must be defined in the [LEVEL2] group.

**CAUTION:** Always use the default settings unless your system/network administrator or NetWeave technical support specifically instruct you to change them.

Keyword	Default	LEVEL2	Description	Platform
TCPIP_BUFFER_SIZE	8192	Yes	The maximum size (in bytes) of an IP packet.	All
TCPIP_QUEUE_DEPTH	10	Yes	The initial estimate of the number of packets of TCPIP_BUFFER_SIZE that NetWeave must accumulate to form the largest expected message.	All
MAX_SEND_SIZE	See Description	No	The number of bytes in the largest frame you expect to send or receive on the connection defined by this group. <b>NOTE:</b> Default=TCPIP_BUFFER_SIZE. MAX_SEND_SIZE must not exceed the TCPIP_BUFFER_SIZE value.	All



Keyword	Default	LEVEL2	Description	Platform
TCPIP_ADDRESS		No	Required. Either a host name or a dotted IP address. Consult your system manager for this value.	All
TCPIP_BACKLOG	4	No	A parameter to the sockets call listen that indicates how many pending calls can be queued while NetWeave answers the current call. This parameter affects the call nwds_ipc_publish.	All
TCPIP_PORT		No	Required. Either a service name or a numeric IP port. Consult your system manager for this value.	All
CLIENT_TCPIP_ADDRESS		No	Bind to a specific address before calling connect. Use this parameter to cross firewalls that filter by IP address and port.	Window Tandem
CLIENT_TCPIP_PORT		No	Use with CLIENT_TCPIP_ADDRESS to specify the port.	Windows Tandem
TCP_NODELAY	0	Yes	If TCP_NODELAY=1, TCP/IP will send a packet immediately. The default is to wait a short interval for additional data.	Windows UNIX Tandem
TCP_LINGER	1	Yes	The TCP/IP closesocket call performs a graceful, non-blocking close. Any queued data is sent (if possible), and the underlying socket is not released until the data has been sent.	Windows
TCP_LINGER_ONOFF	1	Yes	For <i>hard close</i> , set onoff=1 and time=0. The other side gets condition ECONNRESET, and any data waiting to be sent is discarded.	Windows
TCP_LINGER_TIME	0	Yes	Use with TCP_LINGER to specify how long (in seconds) to wait for the non-blocking close to finish.	Windows
TCP_SNDBUF_SIZE	None	Yes	Sets the socket option SO_SNDBUF.	Windows
TCP_RCVBUF_SIZE	None	Yes	Sets the socket option SO_RCVBUF.	Windows
REQUEST_QUEUE_DEPTH	10	No	The initial number of packets NetWeave reserves for storing a complete message.	IBM/ CICS
TRIES	10	No	The number of times TCPA (the router) will try to connect to TCPB (its partner) before returning failure.	IBM/ CICS
TRANSACTION	TCPB	No	The task name of the TCPB partner process.	IBM/ CICS
MAC_TCP_DEVICE	.IPP	Yes	The device name passed to the POpen call.	MacOS



Keyword	Default	LEVEL2	Description	Platform
TCPIP_MAX_EXIT_WAIT	5	Yes	How long (in seconds) NetWeave waits after shutdown for outstanding I/O to complete.	Novell LAN
TCPIP_MIN_EXIT_WAIT	2	Yes	The minimum time (in seconds) that NetWeave waits for I/O to complete.	Novell LAN
TCPIP_INETD_NAME	BG:	Yes	The name passed in the init descriptor in the call to sys\$assign.	Digital
TCPIP_PROCESS_NAME	\$ztc0	No	The name of the TCP/IP process that controls the LAN on which this connection is configured.	Tandem
MAX_RETRY_COUNT	3	Yes	The number of times to retry a write if the error is ENETUNREACH.	Tandem

## BROADCAST Protocol

Broadcast protocol allows an application to send or receive broadcast messages. Any number of applications may receive a broadcast message. Broadcasts are sent and received on a logical channel called a port. The broadcast port is actually the name of the INI file group that contains the parameters that the NetWeave Agent uses for sending and receiving broadcasts.

For a user's application, the broadcast port group specifies a route to a local NetWeave Agent. For example, a sender's broadcast port definition can look like this:

```
[MY_BROADCAST]
NAME=AGENT1::BCAST
```

The broadcast port definition for a receiving application is quite similar:

```
[THEIR_BROADCASTS]
NAME=AGENT2::BCAST
```

When the Agent sends broadcast messages on behalf of the applications located on the same computer where it resides, the Agent is called a *broadcaster*. When it receives broadcast messages on behalf of the applications located on the computer where it resides, the Agent is called a *receiver*. The rest of this section describes the parameters in the Agent's INI file that control the Agent's role as broadcaster and/or receiver.

## Parameters to Control the Rate of Broadcasts

One objective of NetWeave broadcast technology is to reach a maximum stable rate of transmission at which all intended recipients receive all messages. To specify a maximum transmission rate (expressed in milliseconds between transmissions), use the INI file parameter `THROTTLE_INTERVAL`. The larger the `THROTTLE_INTERVAL`, the slower the throughput.



Throttling is dynamic. The broadcast rate cannot exceed the rate derived from `THROTTLE_INTERVAL`, and does not fall below the rate calculated from `MAX_THROTTLE_INTERVAL`. If a recipient detects loss of messages, it sends a special control message back to the Agent that controls the broadcasts. When the sending Agent receives a control message, it reduces its rate of broadcasting by increasing the length of the throttle interval. `THROTTLE_ADJUST` determines the increment by which the throttle interval is increased. If the broadcaster has been sending at a rate below the maximum rate and has not been notified of additional loss of messages, it will gradually increase its transmission rate by lowering the throttle interval back to the user-defined optimum (the original `THROTTLE_INTERVAL` setting).

**NOTE:** If an application tries to flood the network with too many messages at one time, a call to `nwds_ipc_broadcast` returns the error `LEVEL2_WOULD_BLOCK`.

## Parameters to Control Accurate Delivery of Broadcasts

UDP/IP is the basis of NetWeave's broadcast technology. In UDP/IP, a sender broadcasts a message to an undefined collection of receivers. There is no feedback (reverse message flow) from any recipient back to the broadcaster to indicate who is listening, what they hear, or whether they hear the messages completely or in the correct order.

To compensate for these UDP/IP limitations, NetWeave generates several special control messages to provide a small amount of reverse message flow. The broadcaster analyzes any control messages it receives to determine what information was missed. If the broadcaster still has a copy of the missing messages, it retransmits them. However, a receiver that has missed too many messages will stop trying to receive broadcasts and will return the error condition `NWDS_FILTER_ERROR` to the client applications that have registered and are waiting for broadcasts.

To define how and when you want a broadcaster to react to control messages, use the following INI file parameters:

- If one recipient is having trouble, there's a good chance that others are too. When a broadcaster receives the first control message, it briefly suspends its regular transmission to collect the expected flurry of control messages. Use `RECOVERY_INTERVAL` to specify how long the broadcaster must pause.
- While suspended, the broadcaster counts any additional control messages it receives. Use `CYCLE_THRESHOLD` to specify the maximum number of messages to count while waiting. If the `CYCLE_THRESHOLD` is reached during the `RECOVERY_INTERVAL`, the broadcaster extends the length of the pause, resets its count, and increments a cycle counter to indicate that some of the recipients are still having trouble receiving error-free transmissions.
- If the broadcaster reaches the maximum number of recovery cycles specified in the parameter `MAX_CYCLES`, the broadcaster assumes that recovery is not possible for some recipients and resumes broadcasting from where it left off. Recipients who never receive a missed message will eventually sign off and return `NWDS_FILTER_ERROR` to their clients.

## Broadcast Parameters for the [LEVEL2] Group

The following parameters apply to all broadcast ports:

Keyword	Default	Description
---------	---------	-------------

BCAST_MAX_SIZE	UDP_BUFFER_SIZE	The maximum message size (in bytes) that a user intends to broadcast.
BCAST_MAX_WRITE_QUEUE_DEPTH	100	Large broadcast messages are sent as a series of IP datagrams. This parameter specifies the maximum number of datagrams available for buffering broadcasts. When the broadcaster's queue depth reaches this value, a call to <code>nwds_ipc_broadcast</code> returns <code>NWDS_WOULD_BLOCK</code> . <b>NOTE:</b> Because this is not a fatal error, you can retry the write later.
BCAST_QUEUE_DEPTH	100	The maximum number of broadcast messages (not datagrams) that NetWeave can buffer during asynchronous operation.
SENDER_ID	See Description	Each broadcaster must have a unique ID number. If a workstation sends broadcasts, <code>SENDER_ID</code> is <i>required</i> and there is <i>no default value</i> . If a workstation's applications only receive broadcasts, you do not have to set a <code>SENDER_ID</code> .

## Broadcast Parameters for a Protocol Group

The following parameters are used to define a protocol group for each broadcast port:

Keyword	Default	Description
COUNTS_INTERVAL	5	The length (in minutes) of each data collection cycle. See MAX_COUNTS_THRESHOLD below.
CYCLE_THRESHOLD	10	Defines how many control messages the system perceives as too many. If a broadcaster receives more than this number of control messages within a recovery cycle, it starts another recovery cycle.
HEARTBEAT_INTERVAL	5	The time (in milliseconds) that a broadcaster waits for another message to be broadcast before it sends a special message to tell receivers that the broadcast mechanism is still active.
HEARTBEAT_TIMEOUT	15	The time (in milliseconds) that a receiver waits for the broadcaster's heartbeat. If nothing is received from a given sender for this period, NetWeave shuts down the sender's port and notifies all applications that broadcasts have terminated abnormally.
INIT_INTERVAL	100	When a broadcaster starts, it sends a special message (the INIT message) to notify receivers that a new series of broadcast messages is about to start. The INIT_INTERVAL is the time (in milliseconds) that the broadcaster waits after sending the INIT message before it starts a new series of messages. To ensure that the INIT message is the very first message from a new broadcaster to arrive at a recipient, the INIT_INTERVAL should be two to three times larger than the throttle interval. <b>NOTE:</b> The UDP/IP protocol on which NetWeave's resilient broadcast services are built does not guarantee that the messages will be received in the order in which they are sent.
MAX_COUNTS_THRESHOLD	50	The minimum number of control messages that a broadcaster must receive from a given receiver within a data collection cycle to qualify the receiver for a listing on the statistics report. NetWeave collects statistics about control messages to help operators identify which receivers are experiencing problems. At the end of a collection cycle, a trace message indicates which receivers have sent more than this threshold number of control messages.
MAX_CYCLES	5	The maximum number of recovery cycles allowed before a broadcaster exits from the recovery phase and resumes broadcasting where it left off.



Keyword	Default	Description
MAX_THROTTLE_INTERVAL	See Description	The maximum time (in milliseconds) that a broadcaster will wait between sending broadcast messages. MAX_THROTTLE_INTERVAL, which is the inverse of the broadcast rate, determines the minimum rate for broadcasts. If you don't specify a default, the system assigns a default value of four times THROTTLE_INTERVAL.
READ_WINDOW_SIZE	30	To buffer receipt of datagrams and detect any that are missing or out of order, NetWeave creates one read window for each new broadcaster. READ_WINDOW_SIZE is the maximum number of IP datagrams the receiver holds in a read window.
RECOVERY_INTERVAL	50	The maximum time (in milliseconds) a broadcaster waits for receivers to recover missed or out of order messages.
RESEND_INDEX	4	Determines when to request an out of order datagram. The RESEND_INDEX marks how many subsequent messages a receiver reads before it generates a control message back to the broadcaster.
RESEND_THRESHOLD	10 control messages per recovery cycle	A receiver's recovery message asks the broadcaster to resend a single missing datagram. If the number of recovery requests from any receiver reaches the RESEND_THRESHOLD during one phase of recovery, the broadcaster restarts from the earliest saved message.
THROTTLE_ADJUST	None	The number of milliseconds that NetWeave adds to the throttle interval value after recovery cycle(s) have occurred. <b>If you don't set this parameter, throttling will not occur.</b>
THROTTLE_ADJUST_INTERVAL	1000	Controls large-scale adjustments to the rate of message transmission. This interval (specified in milliseconds) defines how often NetWeave checks for recovery cycles. If none have occurred during the last interval, the throttle interval (pause between broadcasts) is decreased by the amount specified in THROTTLE_ADJUST. If there have been recovery cycles, the interval is increased by the THROTTLE_ADJUST amount.
THROTTLE_INTERVAL	See Description	The number of milliseconds a broadcaster waits between transmissions of individual IP datagrams. The default is 10 milliseconds, 100 datagrams per second.
WRITE_RING_SIZE	30	The number of IP datagrams that the broadcaster can recall for recovery purposes, i.e., how many messages a broadcaster saves.

The following broadcast parameters control the direct interface between NetWeave and UDP/IP. These parameters (if present) are specified in each protocol group:

Keyword	Default	Description
UDP_BUFFER_SIZE	1024	The maximum number of bytes that may be sent or received in an IP datagram.
MAX_SEND_SIZE		The maximum number of bytes that can be sent in a single datagram. This value must not exceed UDP_BUFFER_SIZE.
MAX_RECV_SIZE		The maximum number of bytes that can be received in a single datagram. This value must not exceed UDP_BUFFER_SIZE.
UDP_RECVFROM_ADDRESS	None	The IP address from which the receiver reads broadcast messages.
UDP_SENDTO_ADDRESS		The IP address to which a broadcaster sends messages.
UDP_BIND_ADDRESS		The IP address to which broadcasters and receivers both bind. The recommended value is 0.0.0.0.
UDP_PORT		The port number (greater than 512) that NetWeave will use for sending and receiving broadcasts.
TCPIP_PROCESS_NAME	\$ztc0	The Tandem parameter for the TCP/IP process that controls the LAN on which NetWeave will broadcast.



## Sample Files for Broadcasting

The sample configuration on the next page shows the settings for the components illustrated in Figure 3 below.

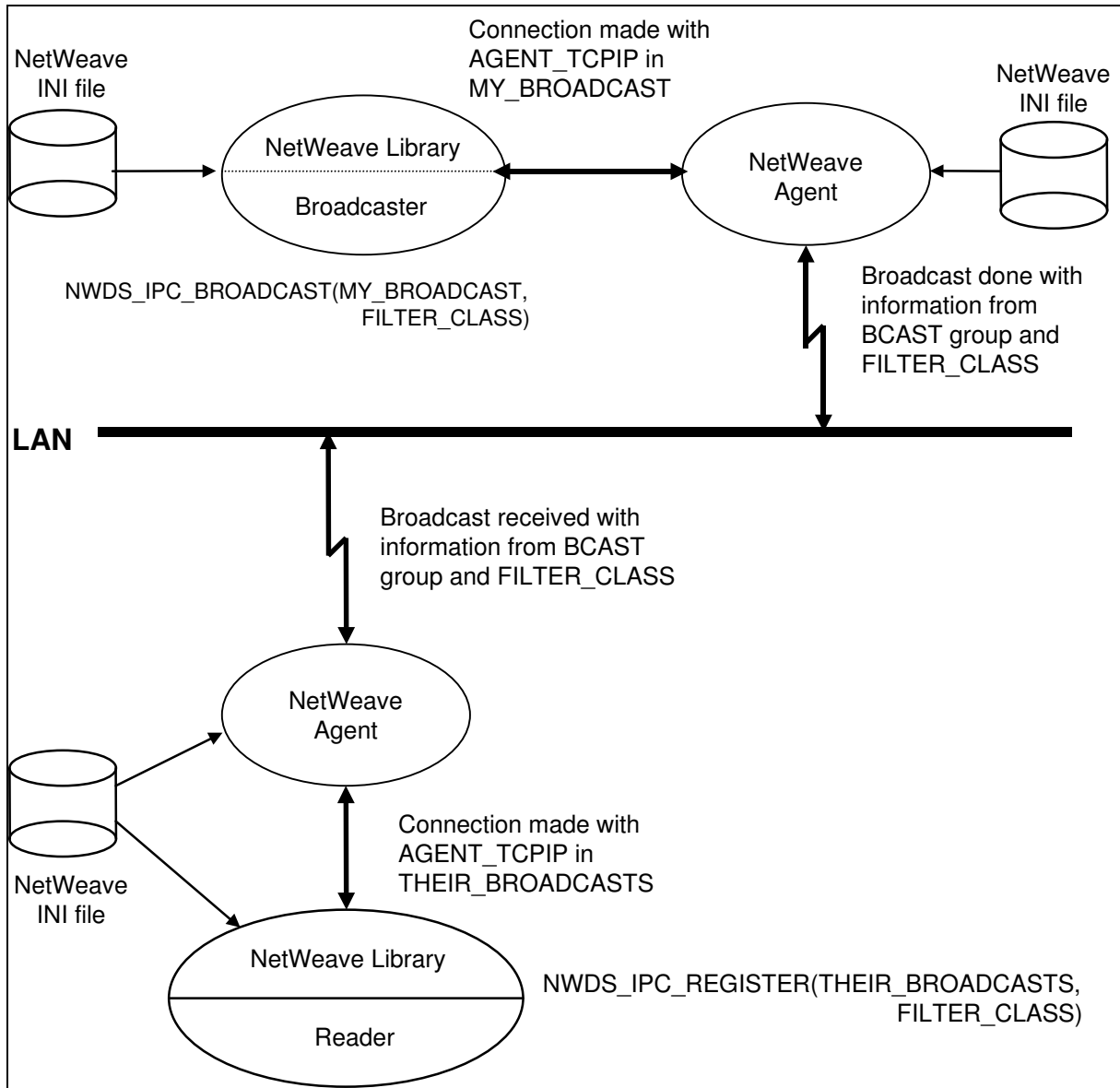


Figure 3. NetWeave's broadcast architecture

## Sender's Sample INI File

The INI file below contains configuration information for an application that generates broadcast messages.

```
[BROADCASTER]
@TRACE_FILE@=<valid file name>
USER_NAME_GROUP=BROADCASTER _PRIVATE

[BROADCASTER _PRIVATE]
<application logical>=<value>

[MY_BROADCAST]
NAME=AGENT_TCPIP::BCAST

[AGENT_TCPIP]
PROTOCOL=TCPIP
TCPIP_ADDRESS=<valid dotted address e.g. 190.130.7.15>
TCPIP_PORT=<valid port number e.g. 7001>
```

## Broadcaster's INI File

The INI file below defines the broadcast parameters for the broadcaster, which is the NetWeave Agent that sends broadcast messages on behalf of the sender applications.

```
[NW_SERVER]
@TRACE_FILE@=<valid file name>
PUBLIC_NAME={AGENT_TCPIP}

[AGENT_TCPIP]
PROTOCOL=TCPIP
TCPIP_ADDRESS=<valid dotted address e.g. 190.130.7.15>
TCPIP_PORT=<valid port number e.g. 7001>

[TRACES]
TRACE_LEVEL=ERRORS
FILE_NAME=@TRACE_FILE@

[BCAST]
LOCAL_PROCESS=1
MAX_SEND_SIZE=1024
MAX_RECV_SIZE=1024
UDP_RECVFROM_ADDRESS=<valid dotted address e.g. 190.130.7.255>
UDP_BIND_ADDRESS=0.0.0.0
UDP_SENDTO_ADDRESS=<valid dotted address e.g. 190.130.7.255>
UDP_BIND_PORT=<valid port number e.g. 14333>
```



```
PROTOCOL=BROADCAST
HEARTBEAT_INTERVAL=5000
HEARTBEAT_TIMEOUT=15000
RESEND_INDEX=4
READ_WINDOW_SIZE=30
WRITE_RING_SIZE=30
XOFF_TIMEOUT=5000
```

## Reader's INI File

The INI file below applies to a client application that receives broadcast messages.

```
[READER]
@TRACE_FILE@=<valid file name>
USER_NAME_GROUP=BROADCASTER _PRIVATE

[READER_PRIVATE]
<application logical>=<value>

[THEIR_BROADCASTS]
NAME=AGENT_TCPIP::BCAST

[NW_SERVER]
@TRACE_FILE@=<valid file name>
PUBLIC_NAME={AGENT_TCPIP}

[AGENT_TCPIP]
PROTOCOL=TCPIP
TCPIP_ADDRESS=<valid IP address of another computer, e.g. 190.130.7.30>
TCPIP_PORT=<valid port number e.g. 7001>

[TRACES]
TRACE_LEVEL=ERRORS
FILE_NAME=@TRACE_FILE@

[BCAST]
LOCAL_PROCESS=1
MAX_SEND_SIZE=1024
MAX_RECV_SIZE=1024
UDP_RECVFROM_ADDRESS=<valid dotted address e.g. 190.130.7.255>
UDP_BIND_ADDRESS=0.0.0.0
UDP_SENDTO_ADDRESS=<valid dotted address e.g. 190.130.7.255>
UDP_BIND_PORT=<valid port number e.g. 14333. NOTE: port numbers must match
across systems.>
PROTOCOL=BROADCAST
HEARTBEAT_INTERVAL=5000
HEARTBEAT_TIMEOUT=15000
```

```

RESEND_INDEX=4
READ_WINDOW_SIZE=30
WRITE_RING_SIZE=30
XOFF_TIMEOUT=5000

```

## DOLLAR\_RECV Protocol (Tandem)

This Tandem-specific protocol is used for the Guardian operating system on Compaq's Himalaya architecture. Use DOLLAR\_RECV protocol for conversations between a NetWeave Agent on Tandem and named server processes that are not controlled by Pathway.

Because a client application on a remote computer cannot use DOLLAR\_RECV protocol, it must use a NetWeave Agent on the Tandem as an intermediary protocol converter. The client application talks to the Agent over TCP/IP, and the Agent relays the messages over DOLLAR\_RECV protocol to the Tandem server process.

The server side uses the standard \$receive mechanism for sending and receiving messages. The Agent uses DOLLAR\_RECV protocol to talk to an existing Guardian process without modifying the server code.

Keyword	Default	Description	Group
TANRECV_QUEUE_DEPTH	10	Initial number of messages that DOLLAR_RECV protocol can send and receive simultaneously.	LEVEL4
NWDS_MAX_RECEIVE_MSG	4096	Maximum length (in bytes) of a message that you can send or receive using this protocol.	
LOCAL_PROCESS=1	1	Required parameter for Guardian protocols. <b>Do not</b> change this value.	Protocol
TANRECV_PROCESS	See Description	The server's Guardian process name.	



## PATHSEND Protocol (Tandem)

This Tandem-specific protocol is used by an Agent to communicate with Pathway serverclasses. PATHSEND protocol provides a good way to interface with existing Pathway serverclasses without having to modify their code.

Keyword	Default	Description	Group
PATHSEND_QUEUE_DEPTH	10	The initial number of messages that PATHSEND protocol can send and receive simultaneously.	LEVEL4
NWDS_MAX_RECEIVE_MSG	4096	Maximum length (in bytes) of a message that you can send or receive using this protocol.	
PATHSEND_PERSISTENT	1	Optional. The default behavior (if you don't specify a PATHSEND_PERSISTENT value) is for a serverclass to stop when there are no more clients connected to it.  If you do specify a PATHSEND_PERSISTENT value, when the last client disconnects from the serverclass, NetWeave returns the error condition LEVEL2_SHUTDOWN. The application decides whether and how to exit.	Protocol
LOCAL_PROCESS	1	Required parameter for Guardian protocols. <b>Do not</b> change this value.	
PATHWAY_MONITOR	See Description	The full Guardian process name of the Pathway monitor where the serverclass is defined.	
PATHWAY_SERVERCLASS	See Description	The name of the serverclass.	



## QACCESS Protocol (IBM/CICS)

In the IBM/CICS environment, NetWeave uses QACCESS protocol for interprocess communications (IPC). QACCESS may use either *temporary storage queues* (TSQs) or *transient data queues* (TDQs). The host application specifies which type of queue to use for a particular IPC task. This section explains how to choose the appropriate queue for IPC in the CICS environment.

TDQs use so-called destructive reads: each message is removed from the queue as it is read. TDQs must be pre-allocated, and are implemented on a disk file. Each logical I/O requires two disk I/Os (one to enqueue a message and another to dequeue it). TDQs are very static; to add or remove a TDQ, you must change the configuration of the CICS region.

In contrast, TSQ reads are not destructive, so the whole queue must be purged to delete the records. Because TSQs reside in memory and not on a disk, you can create and destroy them without affecting the configuration of the CICS region. Use of TSQs by QACCESS is highly dynamic. When QACCESS uses a TSQ to return a reply to a process, the response queue contains only one message and can be safely purged after being read. QACCESS uses the standard CICS ENQ/DEQ facility (a semaphore) to protect a TSQ while it is being drained and purged.

TSQs have two forms:

- QACCESS: the normal, dynamic form that requires no configuration parameters.
- NONE: the older, raw form, which expects all configuration information to be in the INI file. A raw TSQ has one configuration for the program that sends a message and another for the program that reads it. For the sender, the raw TSQ is a *writeq*. For the receiver, it is a *readq*.

Every program that uses NetWeave queues has a single, unique ECA (CICS Timer-Event Control Area) to tell it when a new message appears in one of its queues. The ECA is created on the EXEC CICS POST call. A task waits for a new message by suspending operation on this ECA with the EXEC CICS WAIT EVENT call. Another task may wake up the suspended task with the EXEC CICS CANCEL call. These CICS calls (POST & WAIT EVENT) allow NetWeave to wait for a message efficiently and to wake up a task after sending it a message (CANCEL).

**NOTE:** In an older version of QACCESS protocol, a TSQ was called a Temporary Storage Area, abbreviated TSA. You may occasionally encounter this term, but for our purposes TSA means TSQ.



QACCESS requires the following INI file parameters:

Keyword	Description	Group
ECA_NAME	The ECA to cancel after writing a message on the input queue. There is no default.	WAIT_EVENT
L2QA_QUEUE_DEPTH (default=10)	The number of buffers to preallocate to hold messages within module l2_qaccess.c	LEVEL2
L2QA_BUFFER_SIZE (default=maximum)	The maximum is QIO_BUFFER_SIZE - 1.	
QIO_QUEUE_DEPTH (default=10)	The number of buffers you want to preallocate for holding messages within module cics_qio.c	
QIO_BUFFER_SIZE	Default=maximum=(32763 - 28)	
MAX_SEND_SIZE	Default=maximum=L2QA_BUFFER_SIZE	
SOCKET_NAME	Required. The logical name for this queue, up to 40 characters. There is no default.	Protocol
SOCKET_PROTOCOL	NONE or QACCESS	
SOCKET_READQ_TYPE	TDQ or TSA	
SOCKET_READQ_NAME	The actual queue name.	
SOCKET_READQ_SEMA	The semaphore associated with the read queue (required for TSA).	
SOCKET_WRITEQ_ID_IN_BUFFER (default=0)	0=false; 1=true For dynamic queues, this parameter must be set to true.	
SOCKET_WRITEQ_TYPE	TDQ or TSA	
SOCKET_WRITEQ_NAME	The actual queue name.	
SOCKET_WRITEQ_ECA	The ECA associated with the write queue (required for TSA).	
SOCKET_WRITEQ_SEMA	The semaphore associated with the write queue (required for TSA).	



## Dual-Rail Routing

NetWeave has a fault-tolerant routing methodology known as dual-rail that allows IPC messaging to be distributed over redundant links between applications. Dual-rail routing operates over both TCP/IP and UDP/IP.

Transparent to both sending and receiving applications, messages are simultaneously transmitted over redundant links (dual rails). The NetWeave receiver accepts the first message received and passes it to the application, while the duplicate message received over the alternate channel is silently discarded. If one network link is broken, NetWeave periodically tries to recover the link while continuing to transmit messages over the single remaining link. When the broken link is successfully recovered, message traffic resumes over both channels.

Although dual-rail can be considered a protocol for communications stack purposes, it differs from other protocols in that it appears to be "above" TCP/IP or UDP/IP. To configure a connection for dual-rail, set the connection's protocol as follows:

```
PROTOCOL=DUAL_RAIL
```

After setting the protocol type, you must include the individual rail definitions in the same group where dual-rail is defined. Each rail (connection) points to another group in the INI file that contains the protocol details for that rail.

```
RAIL_1_NAME=first protocol group  
RAIL_2_NAME=second protocol group
```



## Miscellaneous Configuration Issues

### Compatibility With Older Versions of NetWeave

To communicate with another program built with NetWeave version older than 1.5.7, a newer program must set all of its communications to match the older version's settings. To enable a newer NetWeave release to work properly with these old versions, do the following:

To communicate with....	Add the following to the newer NetWeave's INI file
Version 1.5.2 or older	[RPC_CLIENT] 502_ITEM_LIST=1
Versions 1.5.3 through 1.5.6	[RPC_CLIENT] 506_ITEM_LIST=1

To communicate with another program built with NetWeave versions between 1.5.7 and 1.6.x, add the following to the Protocol group:

```
USE_V5=1
```

For all NetWeave versions after 1.7, two programs exchange information during connection setup to negotiate parameters that control their communications.

### SQL Server for Tandem

To improve reliability and scalability of applications that access SQL databases on Tandem computers, NetWeave provides some additional features. If you want to use one logical name to represent multiple NonStopSQL servers, use the CLASS parameter to identify the collection of servers. (This feature is similar to the Load Balancing feature described on page 15.)

The example below identifies a static pool of two SQL servers, NS1 and NS2. The Agent uses NetWeave's DOLLAR\_RECV protocol to communicate with these processes named \$NS1 and \$NS2 respectively.

```
[SQLCONNECT]
CLASS={ NS1, NS2 }
```

```
[NS1]
LOCAL_PROTOCOL=1
PROTOCOL=DOLLAR_RECV
TANRECV_PROCESS=$NS1
@TRACE_FILE@=$S.#NS1
```

```
[NS2]
LOCAL_PROTOCOL=1
```

```

PROTOCOL=DOLLAR_RECV
TANRECV_PROCESS=$NS2
@TRACE_FILE@=$$.#NS2

```

## The NetWeave Dispatcher

A Dispatcher is a special type of server for operating systems that support threads. You can create a Dispatcher on Windows NT or on UNIX.

A Dispatcher must have a unique internal, external, and control publish name. To make a program into a Dispatcher, you must add three groups, similar to the ones shown below, to its INI file. You may name these groups anything you like. The API for `nwds_dispatcher_create(...)` accepts these three names and defines their roles.

```

[INTERNAL]
PROTOCOL=TCPIP
TCPIP_ADDRESS=127.0.0.1
TCPIP_PORT=3102

```

```

[EXTERNAL]
PROTOCOL=TCPIP
TCPIP_ADDRESS=127.0.0.1
TCPIP_PORT=3103

```

```

[CONTROL]
PROTOCOL=TCPIP
TCPIP_ADDRESS=127.0.0.1
TCPIP_PORT=3104

```

## The [Blocking] Group

NWDS file I/O allows more than one record in a file to be read or written during one API call. (For more information about this capability, called record blocking, see the functions beginning with `nwds_file_in` in the *NetWeave API Guide*.) NetWeave item lists and special configuration parameters determine how record blocking works. To read or write several records in a file simultaneously, use `nwds_file_open` with the item `NWDS_FILE_BLOCKING` set to `NWDS_FILE_BLOCKING_ON`.

The [BLOCKING] group defines two parameters that control how records are stored in blocks:

Keyword	Default	Description
BLOCK_TERMINATOR	Carriage Return, ASCII value 10	Delimits records in a block.
BLOCK_SIZE	32567	The number of bytes to hold multiple records. The default value is the maximum value allowed.



## The [FILE\_COPY] Group

The function `nwds_file_copy` sends more than one record at a time to the destination system. The [FILE\_COPY] group controls the size of the buffer that NetWeave allocates for holding multiple records. Use `BUFFER_SIZE` (default=32567) to specify the buffer size for file transfers.

## The [RPC\_CLIENT] Group

NetWeave sends the parameters and values for each API from the client system to the server for execution. The [RPC\_CLIENT] group controls how this information is sent.

Keyword	Default	Description
QUEUE_DEPTH	10	The number of simultaneous messages to process.
BUFFER_SIZE	32767	The maximum message size handled by this module.
502_ITEM_LIST	1	Backward compatibility to NetWeave version 1.05.02.
506_ITEM_LIST	1	Backward compatibility to NetWeave versions 1.05.03 through 1.05.06.

## The [RPC\_SERVER] Group

The [RPC\_SERVER] group is a special group for the NetWeave Agent. It has two parameters, `QUEUE_DEPTH` and `BUFFER_SIZE`, which have the same roles as `RPC_CLIENT`.

## The [LEVEL4] Group

This group controls the efficient delivery of messages. Its `QUEUE_DEPTH` parameter is similar to `RPC_CLIENT`. For larger messages, increase the value of `NWDS_MAX_RECEIVE_MSG` (the maximum is 32K less 200 bytes of NetWeave overhead).

Parameter	Default	Description
QUEUE_DEPTH	10	The number of simultaneous messages to process.
ENCRYPTION_REQUIRED	0	Indicates whether encryption is required. If yes, set <code>ENCRYPTION_REQUIRED=1</code> .
PATHSEND_QUEUE_DEPTH (Tandem only)	10	The number of Pathway messages to process simultaneously.
NWDS_MAX_RECEIVE_MSG (Tandem only)	4096	The maximum message size (in bytes) for Guardian I/O.



## The [NWDS\_QUEUE\_CONTROL] Group

This group contains settings for NetWeave's internal memory queues. Do not override the default settings without consulting NetWeave's technical staff.

Keyword	Default	Description
QUEUE_PADDING	5	Controls the minimum number of elements in a free queue.
QUEUE_INCREMENT	10	Changes the default increment.
QUEUE_DECREMENT	10	Changes the default decrement.
QUEUE_MODULO	10	Controls how often queue increases/decreases are logged.
QUEUE_CHECK	0	Controls queue verification, usually set to false (0).

## Setting the File Type

Normally, the NetWeave item lists control how the API accesses a file. You may set the NetWeave file type parameter in the group that describes the file, and omit it from the item list.

File type	Parameter	Description
C streams	C_FILE=1	Indicates that this is a C File.
FIFO	FIFO_FILE=1	Indicates that this is a FIFO file.



## Hierarchy of INI Files

An initialization (INI) file is a configuration file that a NetWeave application or NetWeave Agent reads at startup. The call to `nwds_init` tells NetWeave which INI file and root group to use. If several programs use the same INI file, you should set up and select a different root (start) group in that INI file for each application.

To organize configuration information that is shared by several applications, you can create a hierarchy or *chain* of INI files, one file linked to the next. The lowest level in the hierarchy is the INI file declared by each application when it calls `nwds_init`. The more broadly applicable the information, the higher in the INI file hierarchy it should reside. If an INI file contains a link to another INI file, this link (the destination INI filename enclosed in curly braces) appears as the last entry of the application's root group.

In the example below, an application has a start group `[START1]` that defines its error log and then points to a higher level INI file called `common.ini` that contains all other configuration information.

```
[START1]
TRACE_FILE=/usr/nwds/start1.err
{common.ini}
```

To translate an alias, NetWeave begins by looking in the root INI file. If it doesn't find the alias, but finds a link to another INI file, NetWeave continues to search the next INI file, and the next, until it either finds and translates the alias, or it runs out of INI files to search. If NetWeave can't find an alias, the translation fails and the NetWeave Agent that called Naming Services returns an error to indicate that the alias could not be found.

Figure 4 shows how two INI files can hold configuration data. The lower level INI file contains configuration information for a single program. The higher level INI file contains global references, such as the identification and routing information for all servers, that are the same for programs and communications protocols within a project.

**NOTE:** A hierarchy of INI files is *optional*, and you may well decide to put all configuration information into one INI file. However, the advantage of hierarchical INI files is that they are much easier to maintain because the global information occurs in just one place, the higher level INI file.

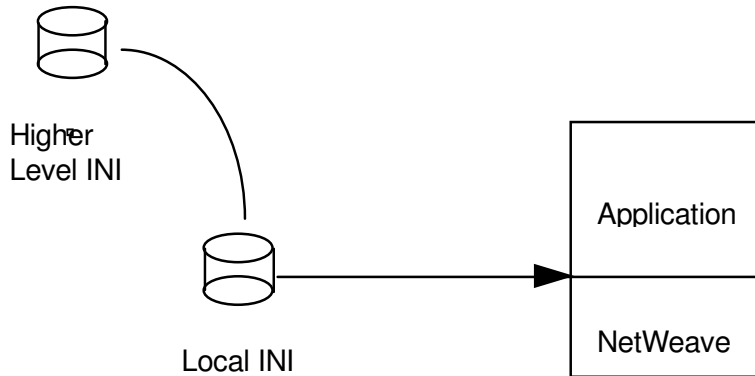


Figure 4. A hierarchy of INI files

The INI file hierarchy may contain both static and dynamic INI files. With dynamic INI files (added in release 2.0) you can add or change certain types of configuration information while a process is running, and NetWeave will incorporate the changes the next time the application checks for this configuration information.

## Dynamic INI Files

Dynamic INI files are an optional way to expand the traditional means of accessing configuration information. (If you don't want to use dynamic INI files, you can configure NetWeave so that your programs continue to access all information from static configuration files.)

INI Server is a NetWeave process that must be started before the NetWeave Agent and before any other processes that will access configuration information dynamically. INI Server has its own static INI file that contains the information required to run INI Server, such as the list of INI files from which INI Server extracts configuration information.

In addition, each NetWeave application also needs an INI file called the Bootstrap INI (or Boot INI) that contains the information that describes how to connect to INI Server. **All configuration information in a Boot INI is static.**

The INI Manager process is a standalone console program that you can use to send commands to an INI Server. For example, after modifying one of the INI files, you can use INI Manager to instruct INI Server to reload the configuration file.

Inside the NetWeave library and in the NetWeave Agent, the functions that retrieve configuration information are synchronous blocking calls. In versions before 2.0, the NetWeave library read the configuration information from the INI file when the application started, and there was no penalty to access this static configuration information using synchronous functions. However, when you use dynamic INI files, an application may be blocked while it retrieves information from INI Server. For example, when you instruct INI Manager to reload an INI file, it may not respond to a program's request for information within a reasonably short period of time. The number of groups in the file and the relationships between them determine how long it takes INI Manager to reload and reprocess the configuration information.

To minimize the impact of reloading on other programs, INI Manager reads part of the new information and then stops to check if another program is requesting configuration information. None of the new information is made available until all modified information is reloaded.

Use the following NetWeave parameters to manage INI Manager's response rate to another program's request:

Keyword	Description
MAX_AMOUNT_PER_READ	Specifies how many bytes INI Manager reads during one reload cycle.
LOAD_PAUSE_MILLISEC	Specifies how long (in milliseconds) INI Manager waits for a program request before continuing to reload information.

## Starting INI Server

INI Server is a command line application that requires three parameters:

- The name of INI Server's own INI file
- The start group in this INI file
- The name by which this INI Server will be known to the other applications

For example:

```
iniserv INI.INI START INI_SERVER
```

## A Sample INI File for INI Server

INI Server's start group must contain a `USER_NAME_GROUP` that points to the other INI files that INI Server must load at runtime. You can call the `USER_NAME_GROUP` whatever you want (in the example below, we call it `SERVER_PARAMS`), but it must contain the parameter `FILE_COUNT` and a list of files for INI Server to load.

`FILE_COUNT` must be at least 1 and may be as large as needed. If `FILE_COUNT` is `n`, there must also be "`n`" entries called `FILE_1`, `FILE_2`, ..., `FILE_N` to specify the logical names of the various INI files that INI Server will load when its starts. Normally, in a separate list you also supply a physical path and filename to map to each logical name. This map may contain more names than `FILE_COUNT`.

Later on, you can use INI Manager to instruct INI Server to load configuration information from one or more of these additional files. When INI Server starts, it will load all of the mapping information even though a particular file may not appear in the `FILE_COUNT` list. Remember, for each logical name that you want INI Server to load at startup, you have to specify the full path and filename in INI Server's Boot INI file.

```
; The INI file's start group.
[START]
@TRACE_FILE@=/usr/ini.err
USER_NAME_GROUP=SERVER_PARAMS
```

```

[SERVER_PARAMS]
;INI files to load when server starts. Specify logical file name.
FILE_COUNT=2
FILE_1=INI_COMMON_1
FILE_2=INI_COMMON_2
;
;list of supported INI files (logical INI file name to physical name
mapping).
INI_COMMON_1  =/usr/common.ini
INI_COMMON_2  =/tmp/common.ini
;
;optional number of milliseconds to wait during INI file load when max read
occurs.
;default is 100 milliseconds.
LOAD_PAUSE_MILLISEC=200
;optional amount of bytes to read from INI file before pausing. Default is
24000.
MAX_AMOUNT_PER_READ=4000

;server's publish group; this name must match the third item in the startup
parameters.
[INI_SERVER]
LOCAL_PROCESS=0
PROTOCOL      =TCPIP
TCPIP_ADDRESS=127.0.0.1
TCPIP_PORT    =11111

*
* Well Known NetWeave Groups
*

[TRACES]
TRACE_LEVEL=ERRORS
MSGLOG_LEVEL=INFO
FILE_NAME=@TRACE_FILE@

```

## A Sample Boot INI for an Application

When an application starts, it reads the parameters in its Boot INI file. Most Boot INI file parameters are static and cannot be changed while the program is running. To maximize the flexibility of your dynamic INI files, keep as little information as possible in the Boot INI files.

If you do want to use dynamic configurations, make sure that an application's Boot INI file includes the group `[DYNAMIC_INI]`. This group must specify the item `LOGICAL_INI_NAME` that represents the lowest INI file in the hierarchy of dynamic INI files. `LOGICAL_INI_NAME` must match one of the logical names in INI Server's Boot INI.

In addition to logical names, the `[DYNAMIC_INI]` group may specify how long the application should continue to use the current configuration information before checking with INI Server for updates. To



help manage the overhead of looking up dynamic information, use `GROUP_EXPIRE_PERIOD` (measured in seconds) to specify how long the INI file group information remains in effect after it is updated from INI Server. The larger the `GROUP_EXPIRE_PERIOD`, the less often the application contacts INI Server to refresh stale information. The default is five minutes.

An application normally requests new information from INI Server when:

- The application does not have the information it needs in its Boot INI file.
- The information is stale (it is older than `GROUP_EXPIRE_PERIOD`).

For each program, the program designer must assess how best to balance the tradeoffs between infrequent checks that minimize the overhead of dynamic information, and frequent checks that increase the flexibility and responsiveness of the application to changes to its environment.

```
[START_HERE]
@TRACE_FILE@=/tmp/netweave.err

[DYNAMIC_INI]
;optional delay before retrying to connect to the INI Server; default 5
seconds
CONNECT_RETRY_PERIOD=10
;optional group expire period; default 300 seconds.
GROUP_EXPIRE_PERIOD=600
;required logical INI file name
LOGICAL_INI_NAME=INI_COMMON
;optional start group name; default is same start group used for bootstrap
file
START_GROUP=START
;optional INI server connectivity group; default is the group name
INI_SERVER
INI_SERVER_GROUP=INI_SERVER

[INI_SERVER]
LOCAL_PROCESS=0
PROTOCOL      =TCPIP
TCPIP_ADDRESS=127.0.0.1
TCPIP_PORT    =11111
*
* Well Known NetWeave Groups
*

[TRACES]
TRACE_LEVEL=ERRORS
FILE_NAME=@TRACE_FILE@
```

## A Sample Common INI File

A common INI file does not contain any required groups or parameters. Normally, this file contains a group whose name is specified by the `START_GROUP` parameter in the Boot INI file's `[DYNAMIC_INI]` group. In the example below, this group is shown as an empty placeholder called `[START]`. To use chained INI files, you must have a start group that contains the reference to the next link in the chain. Chains of INI files are discussed in more detail on page 62.

```
[START]
*
* Local agent
*

[TANDEM]
LOCAL_PROCESS=0
PROTOCOL      =TCPIP
TCPIP_ADDRESS=localhost
TCPIP_PORT    =18475

*
* Logical Filenames
*

[FIFO0]
NAME=TANDEM::FIFO0
[FIFO1]
NAME=TANDEM::FIFO1

*
* Load Balancing
*

[SERVICE_A]
NAME={S1,S2}

[S1]
LOCAL_PROCESS=0
PROTOCOL      =TCPIP
TCPIP_ADDRESS=TANDEM1
TCPIP_PORT    =4001

[S2]
LOCAL_PROCESS=0
PROTOCOL      =TCPIP
TCPIP_ADDRESS=TANDEM2
TCPIP_PORT    =4001

*
* Other agents and processes
*

[HP1]
```

```

LOCAL_PROCESS=0
PROTOCOL      =TCPIP
TCPIP_ADDRESS=HP1
TCPIP_PORT    =18475

```

## Runtime Parameters That You Can Change

Any NetWeave application has the following three parameters that are set a runtime:

- `TRACE_LEVEL` (in the `[TRACES]` group)
- `MSGLOG_LEVEL` (in the `[TRACES]` group)
- `QUEUE_CHECK` (in the `[NWDS_QUEUE_CONTROL]` group)

These parameters are normally static and are specified only in the Boot INI file. However, if a `[TRACES]` group is defined in a dynamic INI file loaded in INI Server, you can change the trace level from `ERRORS` to `FULL` or back without stopping the application. You can also change the level of message logging that your application performs, or enable `QUEUE_CHECK` if NetWeave support staff requests it.

## When Changes Take Effect

INI Server loads its files at startup and reloads them only when instructed to via the INI Manager, as described on page 65. You may make any number of changes to one or more INI files, but until INI Server is instructed to reload them, these changes cannot affect any applications. Even if INI Server has reloaded a changed value, that value will not become known to an application until the application makes a configuration request to INI Server, typically for one of the following reasons:

- The application makes a new IPC connection (by calling `nwds_ipc_publish` or `nwds_ipc_connect`).
- It opens a file or NetWeave FIFO.
- It registers for something using `nwds_ipc_register`, `nwds_trigger_register`, etc.).

As noted above, the setting you choose for `GROUP_EXPIRE_PERIOD` in the `[DYNAMIC_INI]` group controls how often an application returns to INI Server to check for configuration updates. However, these changes are implemented only if:

- You instruct INI Server to reload the (logical) INI file you have changed.
- Your application queries INI Server for updates for stale configuration information.

**NOTE:** An application never needs to request updates to the runtime parameters `TRACE_LEVEL`, `MSGLOG_LEVEL`, and `QUEUE_CHECK`. Changes to these parameters are delivered automatically as soon as an application queries INI Server for any updates to any group.

## Chains of INI Files

Before INI Server was added to NetWeave, chains of INI files were the only way to aggregate common information used by many applications. When INI Server was introduced, the rules for chaining changed. However, if you do not use INI Server, chaining still works the way it always has. The old chaining rules also still apply to files that are linked to the Boot INI files. For these files, to link one INI file to another, the Boot INI's [START] group must include the name of the next file that INI Server consults for configuration information.

For example, assume an INI file named `/tmp/link1.ini` has the following start group:

```
[START_HERE]
{/tmp/link2.ini}
```

NetWeave loads the groups from `link2.ini` with the groups found in `link1.ini`. To add another INI file (`/tmp/link3.ini`) to this chain of INI files, the start group in `/tmp/link2.ini` must look like this:

```
[START_HERE]
{/tmp/link3.ini}
```

If two chained INI files contain a group with the same name, the parameter values in the earliest declaration in the chain take precedence over the later declaration. NetWeave traces along the chain of INI files for the first group that matches a program's request.

Chains of INI files within INI Server differ from the traditional chains only in that names of linked INI files must be logical INI filenames rather than explicit physical filenames. Where a Boot INI links to `/tmp/link2.ini` by physical filename, a common INI file will use a logical name known to INI Server. Because more than one chain may be defined to link common INI files, use names such as `[CHAIN_1]` and `[CHAIN_2]` for the start groups in the links of the distinct chains of INI files.

## Chained File Example

In the example below, two chains are defined to link two different INI files to a third, common INI file. INI Server's Boot INI file contains a `USER_NAME_GROUP` called `[SERVER_PARAMS]` that defines the logical INI files and their mappings to actual files. Only two of the files known to INI Server are explicitly loaded. However, because `INI_COMMON` is in at least one chain, it is loaded as well.

```
[SERVER_PARAMS]
FILE_COUNT=2
FILE_1=INI_CHAIN_1
FILE_2=INI_CHAIN_2
; map of each logical INI filename to a physical file
INI_CHAIN_1=/tmp/netweave/chain1.ini
INI_CHAIN_2=/tmp/netweave/chain2.ini
COMMON_INI=/usr/common.ini
```

The first INI file in chain 1, `INI_CHAIN_1`, has as its start group `[CHAIN1]`:

```
[CHAIN_1]
{COMMON_INI}
```

And the second chain starts in INI file `INI_CHAIN_2` with start group `[CHAIN2]`:

```
[CHAIN_2]
{COMMON_INI}
```

The common INI file needs start groups for `[CHAIN1]` and `[CHAIN2]` only if the chains extend beyond the `COMMON_INI` file. If these groups exist in the common INI file, both may be empty; or they may contain additional references to logical INI files. In this simple example, the first chain contains all the groups in `INI_CHAIN_1` and `COMMON_INI`. The second chain contains all groups from `INI_CHAIN_2` and `COMMON_INI`.

To increase flexibility, you can also consider combining several of the ideas presented above. For example, you can put a `[TRACES]` group and a `[NWDS_QUEUE_CONTROL]` group in the initial INI file in a chain of INI files, and put the server names and FIFOs in the INI files at or near the end of the chain. Then, you can alter the runtime parameters for applications in one chain of INI files without affecting applications linked by another chain. In the sample code shown above, if the `[TRACES]` group is defined in `INI_CHAIN_1`, you can change the tracing or message logging levels for applications that use the start group `[CHAIN1]` without affecting levels in applications that start with `[CHAIN2]`.

**CAUTION:** If you define a `[TRACES]` group or `[NWDS_QUEUE_CONTROL]` group in a Boot INI file and include one or both of these groups in the dynamic INI file that INI Server loaded, INI Server will override your Boot INI values with the values it finds in the dynamic INI file. Therefore, you should place these special groups in at least one of the logical INI files that INI Server loads, and link this INI file to other common INI files. Do not put the special groups directly in the Boot INI file. Instead, set the `LOGICAL_INI_NAME` to one of the INI files that contains them.

## Chained Files and Load Balancing

Place any declarations for load balancing in INI files at or near the beginning of a chain of INI files, and put the names of all servers mentioned in any declaration of load balancing into an INI file near or at the end of the chain. If some of the servers are located on LANs that are physically closer or faster, duplicate their names in the list of servers to increase the probability that they will be selected.

For example, two departments might define chains of INI files that allow mutual sharing of their servers. The first department defines a chain that has a declaration of load balancing for `SERVICE_A` that favors use of server applications reached through its NetWeave Agent, `DEPT1_SERVER`. The second department defines its load balancing to prefer servers on `DEPT2_SERVER`. The common INI file defines the protocol groups for all the departmental Agents and server applications.

For the example below, assume that we use the chains from the previous example and add one more logical INI file named `INI_S1`. This file is accessed by the Agent and the server `S1` located on the file server in the first department (department 1). Assume, too, that `INI_CHAIN_1` contains the following:

```
[SERVICE_A]
NAME={S1, S1, S2}
```

and `INI_CHAIN_2` contains:

```
[SERVICE_A]
NAME={S1, S2, S2}
```

and `COMMON_INI` contains the definitions for `S1`, `S2`, and the Agent in department 1:

```
[S1]
NAME=DEPT1_SERVER::S1
```

```
[S2]
PROTOCOL=.....
```

```
[DEPT1_SERVER]
PROTOCOL=.....
```

and the Agent in department 1 contains the beginning of another chain in its Boot INI file:

```
[NW_SERVER]
PUBLIC_NAME={DEPT1_SERVER}
{INI_S1}
```

```
[DEPT1_SERVER]
PROTOCOL=.....
```

and finally, `INI_S1` contains:

```
[S1]
PROTOCOL=.....
```

## The Dynamic INI Management Interface

In normal operation, INI Server is started before any other NetWeave components and continues to run unattended as long as applications continue to use it. For NetWeave release 2.0, INI Manager is a simple command-line application that performs a limited number of housekeeping tasks that affect INI Server.

INI Manager's primary function is to notify INI Server when one of its common, dynamic INI files has changed. Most of the other functions support this basic function.

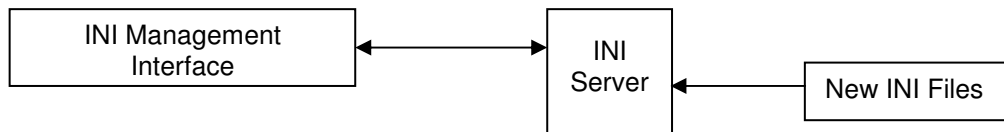


Figure 5. The role of INI Manager

### Starting INI Manager

INI Manager is a command line application that has two runtime parameters: its Boot INI file and its [START] group. This information is passed from the command line to `nwds_init` using the following syntax:

```
inimgmt <INI file> <INI group>
```

Because INI Manager may send commands to any INI Server in the network, the Manager's Boot INI usually contains entries for all INI Servers.

**CAUTION:** Although INI Manager may use dynamic INI files, be very careful when you send commands to the INI Server from which INI Manager receives its configuration information.

### Example 1

Please note that in the following two examples, user input is shown in **bold**.

When the Manager starts, it prompts for the INI Server to which it connects. In this example we are connecting to the INI Server defined by the group [INI\_SERVER]:

```
Only enter the first letter of a command
Enter Dynamic INI Server Connection Group or <E>xit: INI_SERVER
connecting to server (INI_SERVER)
```

After connecting to INI Server, INI Manager prompts for commands until you exit. In this example, we are selecting the Help command to remind ourselves what each function does.

```

Enter:
  <L>oad File, <S>hutdown, <D>isconnect,
  <F>ile Select, <G>roup Fetch, <E>xit, <H>elp: H
Enter first letter of the command:
L - For Server to load/reload an INI file
S - To Shutdown the Dynamic INI Server
D - To disconnect from the Dynamic INI Server
F - Select INI File and Start Group for Group Fetch
G - Fetch group contents from selected INI File
E - To exit this program

```

The most commonly used function is Load File. In this example, we tell INI Server to reload INI\_COMMON\_1:

```

Enter:
  <L>oad File, <S>hutdown, <D>isconnect,
  <F>ile Select, <G>roup Fetch, <E>xit, <H>elp: L
File Select. Select Logical INI File:
1  INI_COMMON_1=/usr/netweave/common1.ini
2  INI_COMMON_2=/usr/netweave/common2.ini
3  INI_COMMON_3=/usr/netweave/common3.ini
Enter A Number or Logical Name: 1
INI File Load Successfully Issued.

```

## Example 2

The functions File Select and Group Fetch display the information that INI Server is sending to applications. In this example from the Tandem, we view the current interpretation of the logical name FIFO0 taken from the file INI\_COMMON. Note that INI Manager input is not case-sensitive.

```

Enter:
  <L>oad File, <S>hutdown, <D>isconnect,
  <F>ile Select, <G>roup Fetch, <E>xit, <H>elp: F
File Select. Select Logical INI File:
1  INI_COMMON=$lib15.mf200.comnini
Enter A Number: 1
File Select. Enter Start Group: start
INI File Select Successfully Issued

Enter:
  <L>oad File, <S>hutdown, <D>isconnect,
  <F>ile Select, <G>roup Fetch, <E>xit, <H>elp: g
Fetch Group. Enter Group Name: fifo0
Fetch Group Successful. Group data:
*****start*****

```





```
LOAD TIME=34 14:54
```

```
QUEUE_CHECK=1  
TRACE_LEVEL=ERRORS  
MSGLOG_LEVEL=TRACE
```

```
[FIFO0]  
NAME=TANDEM: :FIFO0  
*****end*****
```

The Load Time is the Julian day, hour and minute when the INI file called `INI_COMMON` was loaded in INI Server. Next, the display of group data shows the current values of the alterable runtime variables. Finally, the group name and contents are listed.

To break the connection to INI Server gracefully, use the command `Disconnect`. (Using the `Exit` command severs the connection automatically.) When you disconnect from one INI Server, you will be prompted to either exit or connect to another INI Server. There is no separate command for connecting to an INI Server.

**NOTE:** Please be careful when using the `Shutdown` command, because this shuts down all links between INI Server and applications that are attached to it.



## Message and Error Logging Considerations

There are three ways to control the level and type of message logging from NetWeave components and applications:

- Basic error tracing
- Application message logging
- Platform-specific logging

Generally, in researching a connectivity problem, the more information you have the better. However, having maximum logging on all the time is very expensive in terms of disk space and performance.

### Basic Error Tracing

The NetWeave Tracing feature, described on page 8, includes the following basic components configured within the [TRACES] group:

- A filename for log output (`TRACE_FILE`).
- The level of output information (`TRACE_LEVEL`) you want to include in the log.
- A parameter for maintaining a reusable cache of detailed information (`TRACE_SAVE`) that is included in the log file when an error occurs. Use the `SAVE_LINE_COUNT` parameter to specify how many lines should be stored in this cache.

Typically, each application running on a platform will generate traces to its own log file, although it is possible – but confusing – to generate all of the traces to a common file. You can specify a `TRACE_FILE` value for each application using either unique INI files, or (a better solution) using a macro substitution such as `@TRACE_FILE@` to differentiate the trace file name through different root group definitions.

For threaded platforms, output from each thread is identified in the log, but the trace settings apply to all threads in the process. The thread number, enclosed in parentheses, is shown in the log file after the INI file name.

With the release of NetWeave 1.7, all messages related to a particular event (usually an error) are presented in an error “basket” to help identify the problem. A basket usually consists of several lines plus a single banner line that identifies the date/time, INI file, thread number (if applicable), and severity of the event.

**NOTE:** To preserve the serial nature of the information in the log file, error basketing is disabled when `TRACE_LEVEL` is set to anything other than `ERRORS`.

## Application Message Logging

NetWeave applications can use the `nwds_msglog` function to write application-specific information to the NetWeave message log. For more information about `nwds_msglog`, see the *NetWeave API Guide*.

As noted above in section Message Log Facility, the NetWeave [TRACES] group contains the following optional parameters that applications may use with `nwds_msglog` calls:

Keyword	Description
MSGLOG_LEVEL	Determines the level of information that appears in the log file for <code>nwds_msglog</code> calls. For example, if MSGLOG_LEVEL is set to ERROR, calls to <code>nwds_msglog</code> with severity NWDS_MLSTRACE are not included in the log file.
MSGLOG_ID	Provides another way to identify applications in the MVS/CICS platform-specific logging facility (see below).
TRACE_APPL_FLUSH (default=0)	Use this parameter to integrate application-level messages in the NetWeave error baskets in the log file. If TRACE_APPL_FLUSH=0 (or not specified), user messages and error baskets will appear as independent messages.  <b>NOTE:</b> If TRACE_APPL_FLUSH=1, and if the application does not call <code>nwds_msglog</code> when an error code is returned to NetWeave, no error information at all is written to the log. Therefore, when you investigate a problem that does not seem to be reflected in the log file, always rerun the test with TRACE_APPL_FLUSH=0 to ensure that errors are not being discarded because the TRACE_APPL_FLUSH flag is not set properly.

## Platform-Specific Logging

NWDS Version 2.0 includes platform-specific (PS) logging that is easier to integrate with any existing tracing mechanisms. This feature generates a one-line error message to a platform-specific report device. For more information about the error, you should always review the NetWeave log files. Please note that the platform-specific features *augment*, not replace, the existing functions.

The following conditions determine which one-line message is sent to the platform-specific log:

- If an application is not using `nwds_msglog()`, then the one-line error is the core error returned from the NetWeave core O/S layer. The error code in this message would most likely be an O/S-defined error code from a file such as `errno.h` on UNIX or `winerror.h` on NT.
- If an application is using `nwds_msglog()`, but TRACE\_APPL\_FLUSH is not set, or set to 0, then any application messages with severity at least NWDS\_MLSEERROR (and the associated NetWeave core O/S layer messages) are written to the platform-specific log.

- If an application is using `nwds_msglog()` and `TRACE_APPL_FLUSH` is set to 1, then the application's messages with severity at least `NWDS_MLSEERROR` are sent only to the platform-specific log.

To enable platform-specific logging, set `TRACE_SYSLOG_ENABLED=1` on platforms that support PS logging. Platforms that do not support PS logging will ignore this switch. The following sections explain how platform-specific logging works on each platform for which it is currently available.

## Windows NT

Platform-specific logging on WinNT sends messages to the Windows NT Application Event Log. To look at these messages, use the Event Viewer in the Administrative Tools folder. Because WinNT requires that a Registry key be present for each application that generates messages, an application must register a messages file that dictates how to interpret the messages in the event log. To assist this process, the NetWeave NT distribution contains two files, `nwreglog.exe` and `nwmsgs.dll`, that must be present and executed on each machine that collects NetWeave PS log messages. To run `nwreglog`, do one of the following:

- If `nwreglog.exe` and `nwmsgs.dll` are in the same directory, simply execute `nwreglog`.
- If `nwreglog.exe` and `nwmsgs.dll` are in different directories, execute `nwreglog <PathtoMsgsfile>`, where the parameter indicates the pathname of the `nwmsgs.dll` file.

**NOTE:** Because `nwreglog` does not run over a telnet session, it must be run from the console of the Windows NT machine.

If a network contains multiple NT machines, you may consolidate the platform-specific log messages from one machine's event log by specifying the machine name as the value of the parameter `PS_NT_LOG_SERVER`. Note that the machine on which the messages are to be logged and viewed is the machine that requires the execution of `nwreglog` as described above.

In the example below, the application event log on `MACHINEB` contains the PS messages from the application, and `nwreglog` must be executed on `MACHINEB`:

```
[TRACES]
TRACE_LEVEL=ERRORS
MSGLOG_LEVEL=ERROR
TRACE_SYSLOG_ENABLED=1
PS_NT_LOG_SERVER=MACHINEB
FILE_NAME=@TRACE_FILE@
```

## MVS/CICS

If you are using platform-specific logging on the MVS/CICS platform, `EXEC CICS WRITE OPERATOR` writes messages to the CICS SYSLOG. The following parameters have been defined in the [TRACES] group for operating on MVS/CICS with `TRACE_SYSLOG_ENABLED=1`:

Keyword	Default	Description
CICS_DATE_FMT	YMD	Specifies position of year, month, day in the timestamp. Set to either YMD, DMY, or MDY.
CICS_PROG_ID	NETWEAVE	The Program ID shown after the date/time, before the log text.
CICS_ACTION	NONE	The operator action level issued in the <code>EXEC CICS WRITE OPERATOR ACTION(xx)</code> command. The following text strings are valid values: <ul style="list-style-type: none"> <li>NONE (corresponds to 0)</li> <li>IMMEDIATE (2)</li> <li>EVENTUAL (3)</li> <li>CRITICAL (11)</li> </ul> To use another numeric value, enter the decimal value (either string or decimal).

Sample group from the Router's INI file:

```
[TRACES]
TRACE_LEVEL=ERRORS
MSGLOG_LEVEL=ERROR
TRACE_SYSLOG_ENABLED=1
CICS_DATE_FMT=DMY
CICS_PROG_ID=ROUTER
CICS_ACTION=EVENTUAL
FILE_NAME=@TRACE_FILE@
```

## Tandem

With `TRACE_SYSLOG_ENABLED` on Tandem/Guardian, NetWeave generates PS log messages to the Event Management System (EMS). All PS error messages are logged as critical events.

## UNIX

On all NetWeave UNIX platforms, including RS6000/AIX, DEC/Tru64, Sun/Solaris, HP/Hp-UX, and Linux, the `syslog()` call is invoked with messages destined for the PS log. Depending on the configuration of the `syslogd` daemon, the messages may be forwarded to another host, displayed on the console, or simply stored in the `syslog` file (whose location is platform-dependent).



## Supported Platforms

Platform	Operating systems	Protocols	Default INI file name
DEC	VMS OPEN_VMS	TCP/IP UDP/IP	nwds.ini
PC	Windows 95/98 Windows NT Windows 2000 OS/2	TCP/IP UDP/IP	nwds.ini
<Many>	UNIX	TCP/IP UDP/IP	nwds.ini
TANDEM	Guardian	TCP/IP UDP/IP Dollar_RECV Pathsend	nwdsini
UNISYS	MCP	TCP/IP	nwds.ini
IBM	MVS/CICS	TCP/IP QACCESS	nwdsini

## Sample INI Files

The following sections contain sample INI files that are correct for most platforms. To include comments in an INI file, use the asterisk (\*) at the beginning of a line.

### Simple Communication Between Two Processes

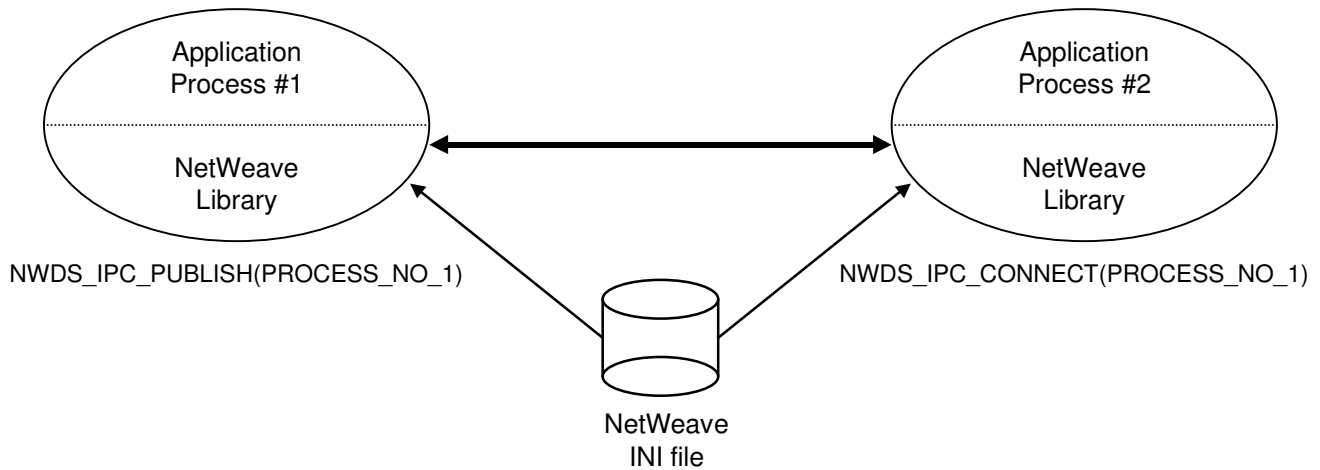


Figure 6. Simple communication between two processes

The INI file below contains the information for application processes #1 and #2:

```
[PROCESS_NO_1]
@TRACE_FILE@=<valid file name>
PUBLIC_NAME={PROCESS_NO_1_TCPIP}
USER_NAME_GROUP=PROCESS_NO_1_PRIVATE

[PROCESS_NO_1_PRIVATE]
<application logical>=<value>

[PROCESS_NO_2]
@TRACE_FILE@=<valid file name>
USER_NAME_GROUP=PROCESS_NO_2_PRIVATE

[PROCESS_NO_2_PRIVATE]
<application logical>=<value>

[PROCESS_NO_1_TCPIP]
PROTOCOL=TCPIP
TCPIP_ADDRESS=<valid dotted address e.g. 190.130.7.15>
TCPIP_PORT=<valid port number e.g. 7001>

[TRACES]
TRACE_LEVEL=ERRORS
FILE_NAME=@TRACE_FILE@
```

## NetWeave FIFO Queues

One process (the producer) sends messages to a second (the consumer).

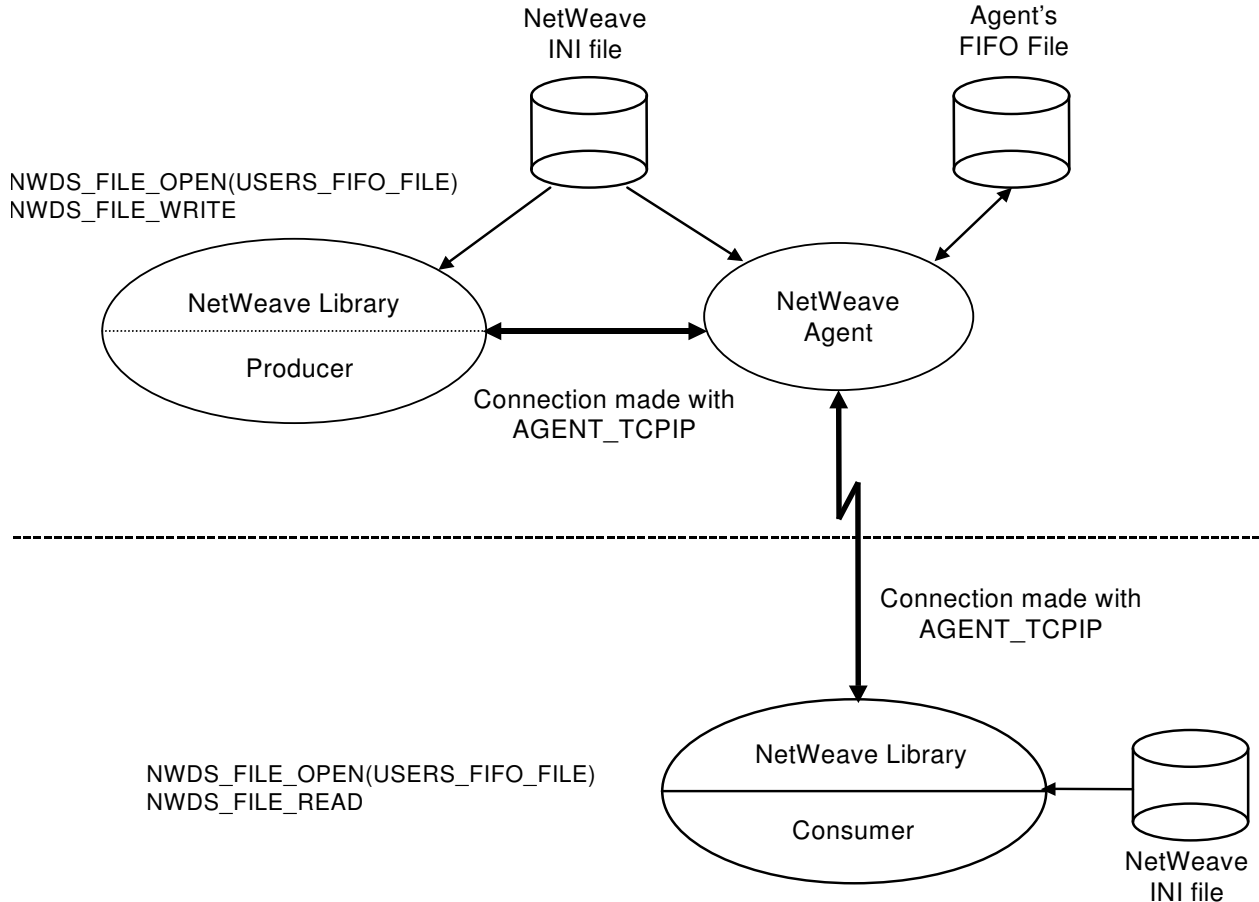


Figure 7. Using FIFO queues to send messages

### Producer's INI File

This sample file assumes that the NetWeave Agent and the FIFO that it manages are co-located with the producer application. In this configuration, the producer application is not affected by network failures. Assume the consumer application is on another computer.

```
[PRODUCER]
@TRACE_FILE@=<valid file name>
USER_NAME_GROUP=PRODUCER_PRIVATE
```

```
[PRODUCER_PRIVATE]
<application logical>=<value>
```



```

[USERS_FIFO_FILE]
NAME=AGENT_TCPIP::AGENTS_FIFO_FILE

[NW_SERVER]
@TRACE_FILE@=<valid file name>
PUBLIC_NAME={AGENT_TCPIP}

[AGENT_TCPIP]
PROTOCOL=TCPIP
TCPIP_ADDRESS=<valid dotted address e.g. 190.130.7.15>
TCPIP_PORT=<valid port number e.g. 7001>

[AGENTS_FIFO_FILE]
NAME=<valid file name>
FIFO_FILE=1

[TRACES]
TRACE_LEVEL=ERRORS
FILE_NAME=@TRACE_FILE@

```

## Consumer's INI File

```

[CONSUMER]
@TRACE_FILE@=<valid file name>
USER_NAME_GROUP=CONSUMER _PRIVATE

[CONSUMER _PRIVATE]
<application logical>=<value>

[USERS_FIFO_FILE]
NAME=AGENT_TCPIP::AGENTS_FIFO_FILE

[AGENT_TCPIP]
PROTOCOL=TCPIP
TCPIP_ADDRESS=<valid dotted address e.g. 190.130.7.15>
TCPIP_PORT=<valid port number e.g. 7001>

[TRACES]
TRACE_LEVEL=ERRORS
FILE_NAME=@TRACE_FILE@

```





## Glossary

Agent	The NetWeave process that controls all input and output to queues, sends notifications to clients when data base changes have occurred, and is responsible for all aspects of security and data conversion.
Asynchronous	An operation in which the applications program is allowed to continue execution while the operation is performed. The access method informs the application program when the operation is completed.
Broadcast services	Simultaneous transmission of data to more than one destination: one sender, unlimited receivers. Message deliveries are connectionless and unacknowledged.
Client-database services	Allows all other computers in the network, regardless of platform type, to access one computer's file system.
Client-server model	A client application sends a request message to a server program. The server program retrieves information or updates a local database on behalf of the (remote) client application.
Client-transaction services	Applications where programs communicate and synchronize operations by exchanging messages (IPC). They are used to implement on-line transaction processing and high-speed, real-time process control applications.
Consumer process	An asynchronous procedure that is responsible for processing the data in a message queue.
Dispatcher	In a distributor-based threaded server, the Dispatcher (provided by Netweave as part of the <code>nwds_dispatcher_</code> function set) is responsible for creating application threads and passing messages to them once started.
Distributor	A NetWeave-provided facility for multi-threaded server processes. The Distributor starts and manages simple application threads for processing messages.
Event-driven design	A non-procedural methodology of software development that is asynchronous in nature, and is fundamentally multi-threaded because it allows you to maintain multiple concurrent sessions.



Interprocess communication (IPC)

The process by which programs communicate data to each other and synchronize their activities.

Item list

A variable-length array of parameters whose last element is a unique type, `NWDS_END_OF_LIST`. Each element (item) in the array has three components:

- *Type*: a constant from `netweave.h` that identifies a parameter (parameter name).
- *Length*: the length of the parameter value. Most parameters are either 16-bit integers (`NWDS_SHORT`) or 32-bit integers (`NWDS_LONG`). Variable-length parameters are considered to be of type `NWDS_CHAR`. For return item lists, the length is the maximum number of bytes that can be copied to the destination location.
- *Pointer to value*: for a control item list, this is the address of the location in memory where you have stored the value you want to assign to the parameter. For a return item list, this is the address in which to store the returned value.

Legacy application

The vast collection of commercial and scientific applications written since the late 70s that share one or more of these features:

- The application resides on a single hardware platform.
- The user interface is the traditional character-oriented terminal.
- Access to related application functions is via menus and function keys.
- Application data are stored in record-oriented files.
- Access to these records is typically through keys and indices.

Loopback testing mode

Used for unit testing locally. Most applications except client-database can (and should) be constructed to run on a single platform. For example, if you are doing IPC messaging, construct a simple client or server to interact with your application. Such a test bed is said to run in “loopback” mode.

Netweave.h

NetWeave header file. Contains the official definition of the API.

On-line transaction processing (OLTP)

A system that processes multiple transactions concurrently and where the data flows to/from the computer directly from the point of origin.

Peer-to-peer model

Data communications between two nodes(processes) that have equal status in the interchange. Any peer node can both generate messages to other processes as well as receive (*unsolicited*) messages from other processes.



Polling for a completion	Monitoring a flag that the completion function sets up when the (asynchronous) NetWeave function finishes.
Producer application	In FIFO message queuing, a producer puts messages at the tail of the queue, and a consumer gets messages from the head of the queue.
Queuing services	NetWeave services that store messages awaiting delivery. Queuing services are often the core of store and forward applications.
Receiver application	A process that reads and reacts to broadcast messages.
To scale (growth of application)	To enlarge or expand either a process, or the number of messages that a process can handle.
Sender application	An application program that generates a message to broadcast.
Synchronous function call	Initiated by a process that requests a specific event. All other processing is suspended until a response is received for the request.
Thread, boss thread, worker thread	The boss/worker thread model is a thread-based mechanism for work distribution between threads. A unit of work is delivered to the boss, which chooses a worker thread to perform the task and then return the result to either the boss or the originator.
UDP datagram	User Datagram Protocol (UDP) is an IP protocol. Datagrams are ideal for broadcasts because they are delivered to the IP network layer regardless how many nodes in the network may consume the information. A datagram is the basic unit of information passed across the Internet environment. It contains a source and destination address along with the data. An Internet Protocol (IP) datagram consists of an IP header followed by the data.
Unsolicited message	A message that a process receives without any prior prompting.
Workflow model	The automobile assembly line is a paradigm for the workflow model in manufacturing. Each cell accepts the outputs of its predecessors as its inputs, modifies the assemblage and passes its output to its successors.