



NetWeave API Guide

User's Guide for Version 2.0 January 2003



www.netweave.com

Copyright © 2002-2003 NetWeave Integrated Solutions, Inc.. All rights reserved.

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

Windows NT 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

INTRODUCTION.....	1
The NetWeave APIs	3
Basic API.....	3
The API Options	5
What's New for Version 2.0.....	6
PROGRAMMING CONCEPTS.....	7
Callback Routines (NWDS_CALL_BACK).....	8
The Item List (NWDS_ITEM_LIST).....	10
INI Files	11
Parameters and Pointers	12
Messaging Limits	13
API FUNCTIONS GROUPED BY USE.....	14
ALPHABETICAL LISTING OF FUNCTION CALLS	17
NWDS_BATCH.....	18
NWDS_CONVERT_DATA.....	20
NWDS_CONVERT_RECORD.....	23
NWDS_DISPATCHER_CREATE.....	27
NWDS_DISPATCHER_STATS	30
NWDS_DISPATCHER_STOP.....	32
NWDS_ERROR_TEXT.....	34
NWDS_EXECUTE.....	36
NWDS_EXIT.....	41
NWDS_FILE_CLOSE.....	43
NWDS_FILE_COPY.....	46
NWDS_FILE_CREATE.....	49
NWDS_FILE_DELETE.....	52
NWDS_FILE_INFO.....	55
NWDS_FILE_OPEN.....	59
NWDS_FILE_POSITION.....	63
NWDS_FILE_READ	67
NWDS_FILE_REMOVE.....	70
NWDS_FILE_UPDATE.....	73
NWDS_FILE_WRITE.....	76
NWDS_INI_DELETE_NAME.....	79

NWDS_INI_GET_INT.....	81
NWDS_INI_GET_NAME.....	83
NWDS_INI_PUT_NAME.....	86
NWDS_INIT.....	88
NWDS_IPC_ACCEPT.....	89
NWDS_IPC_BROADCAST.....	92
NWDS_IPC_CONNECT.....	95
NWDS_IPC_OPTIONS.....	98
NWDS_IPC_PUBLISH.....	101
NWDS_IPC_READ.....	104
NWDS_IPC_REGISTER.....	107
NWDS_IPC_SHUTDOWN.....	110
NWDS_IPC_WRITE.....	113
NWDS_ITEM_LOAD_CHAR.....	116
NWDS_ITEM_LOAD_HANDLE.....	118
NWDS_ITEM_LOAD_LONG.....	120
NWDS_ITEM_LOAD_SHORT.....	122
NWDS_LOGOFF.....	124
NWDS_LOGON.....	126
NWDS_MSGLOG.....	127
NWDS_PASSWORD.....	130
NWDS_PING.....	131
NWDS_SESSION_CLOSE.....	133
NWDS_SLEEP.....	136
NWDS_SLEEP_CALLBACK.....	138
NWDS_SLEEP_CLEAR_CALLBACK.....	141
NWDS_SQL_COLUMN_BIND.....	143
NWDS_SQL_COLUMN_COUNT.....	144
NWDS_SQL_COLUMN_GET.....	146
NWDS_SQL_COLUMN_INFO.....	149
NWDS_SQL_CONNECT.....	152
NWDS_SQL_DISCONNECT.....	154
NWDS_SQL_EXECUTE.....	155
NWDS_SQL_FETCH.....	158
NWDS_SQL_SELECT.....	161
NWDS_STOP.....	164
NWDS_SYSTEM_TYPE.....	166
NWDS_TIMER_START.....	169
NWDS_TIMER_STOP.....	172

NWDS_TP_ABORT	174
NWDS_TP_COMMIT.....	175
NWDS_TP_RESUME.....	176
NWDS_TP_START	178
NWDS_TP_STATUS.....	181
NWDS_TRIGGER_CANCEL.....	182
NWDS_TRIGGER_READ	184
NWDS_TRIGGER_REGISTER.....	187
ITEM TYPES AND VALUES.....	189
Common Item List Definitions	191
Assigning a Constant Length Value to a Parameter.....	191
Assigning a Variable Length Value to a Parameter.....	191
Message Queue (FIFO) Files.....	193
Generic C Files.....	194
NETWEAVE KERNEL FUNCTIONS FOR WINDOWS NT	195
NWDS_NT_CLEAR_EVENT	197
NWDS_NT_DEFINE_EVENT	199
NETWEAVE KERNEL FUNCTIONS FOR UNIX.....	200
NWDS_UX_CLEAR_EVENT.....	202
NWDS_UX_DEFINE_EVENT.....	204
NETWEAVE KERNEL FUNCTIONS FOR DEC, VMS, AND OPENVMS.....	205
NWDS_VMS_CLEAR_EVENT	207
NWDS_VMS_DEFINE_EVENT	209
NETWEAVE KERNEL FUNCTIONS FOR TANDEM.....	210
NWDS_KERNEL_CALL_BACK.....	212
NWDS_KERNEL_RECV_CALL_BACK	214
NWDS_TANDEM_CLEAR_EVENTS.....	218
NWDS_TANDEM_CLEAR_SYSTEM_EVENTS	220
NWDS_TANDEM_DEFINE_EVENT.....	222
NWDS_TANDEM_DEFINE_SYSTEM_EVENT	224
NWDS_TANDEM_REPLYX.....	226
NWDS_TANDEM_RECEIVEINFO	228
RETURN CODES AND RECOVERY.....	229
RETURN CODE NUMERIC DEFINITIONS	238
GLOSSARY	246



Introduction

The NetWeave API provides function calls for client-transaction applications, messaging services, and data server applications. The function calls connect local applications and/or file systems to remote applications and/or file systems.

The NetWeave API is the programmatic interface to the NetWeave product. It is virtually identical on every platform, though there are constructs, known as *itemlists*, which you can use to access platform-specific features through the NetWeave API. The function calls that comprise the NetWeave API are simple to understand and use in their basic form, while allowing enough power and flexibility for industrial strength conditions. Examples of this power and flexibility are the itemlist capability, as well as the ability to use the NetWeave services either synchronously or asynchronously.

The simplest way to use the API is to make synchronous calls. When you call a function synchronously, the operation (such as reading a record from a remote file, or sending a message to a remote application and waiting for confirmation) is performed to completion, and the completion status of the function is returned. The call always waits until the function is completed before it returns to the caller.

In asynchronous operation, the caller specifies a *callback* routine for NetWeave to invoke once the operation is complete, the function calls return immediately with a status of PENDING, and the final operation status is supplied to the callback function that NetWeave will call when the entire operation has completed.

This section introduces some of the basic concepts used throughout the NetWeave API. The rest of the manual lists each NetWeave function call alphabetically, explaining the purpose, parameters, return values, and error codes that are associated with the call.





The NetWeave APIs

Basic API

The basic NetWeave API implements the features of the NetWeave product on behalf of the calling application. The table below lists the basic API functions. For a description of what each function does, please see page 14.

NetWeave option	Function	Used for
Basic NetWeave	nwds_batch nwds_execute	Process initialization
	nwds_init nwds_item_load_char nwds_item_load_handle nwds_item_load_long nwds_item_load_short	Initialization
	nwds_ini_delete_name nwds_ini_get_name nwds_ini_put_name	Retrieving application-specific information from the INI file.
	nwds_ping nwds_system type	Identification
	nwds_tp_abort nwds_tp_commit nwds_tp_resume nwds_tp_start nwds_tp_status	Transaction processing
	nwds_exit nwds_stop nwds_sleep nwds_sleep_callback nwds_sleep_clear_callback nwds_timer_start nwds_timer_stop	Timing/sleep
	nwds_convert_data nwds_convert_record	Translation
	nwds_logon nwds_logoff nwds_password	Security

(continued)



NetWeave option	Function	Used for
Basic NetWeave (continued)	nwds_session_close nwds_ipc_accept nwds_ipc_connect nwds_ipc_options nwds_ipc_publish nwds_ipc_read nwds_ipc_shutdown nwds_ipc_write	IPC services
	nwds_dispatcher_create nwds_dispatcher_stats nwds_dispatcher_stop	Thread services
	nwds_error_text nwds_msglog	Error messages and logging



The API Options

To expand your repertoire beyond what the basic NetWeave API offers, you can add one or more of the other API options:

- Data server: remote file access
- Message queue: guaranteed delivery messaging FIFOS
- Broadcast services: multicast messaging
- RFT: reliable file transfer

NetWeave option	Function	Used for
Data server (formerly client-database)	nwds_file_close nwds_file_create nwds_file_delete nwds_file_info nwds_file_open nwds_file_position nwds_file_read nwds_file_remove nwds_file_update nwds_file_write	File services
	nwds_sql_column_bind nwds_sql_column_count nwds_sql_column_get nwds_sql_column_info nwds_sql_connect nwds_sql_disconnect nwds_sql_execute nwds_sql_fetch nwds_sql_select	SQL
	nwds_trigger_cancel nwds_trigger_delete nwds_trigger_read nwds_trigger_register	Notification
Message queues (FIFO)	nwds_file_close nwds_file_create nwds_file_info nwds_file_open nwds_file_position nwds_file_read nwds_file_remove nwds_file_write	Guaranteed delivery messaging
Broadcast	nwds_ipc_broadcast nwds_ipc_register	Multicast messaging
File transfer	Command line executable	File services
	RFT	API in development
	nwds_file_copy	File services



What's New for Version 2.0

Many of the enhancements for Version 2.0 of NWDS are configuration-related, and do not affect the API. The table below lists the topics that have enhancements and the NWDS document that describes the changes.

Feature	Document
Dynamic INI files	Configuration Guide
INI file management	Configuration Guide
Load balancing	Configuration Guide
Platform-specific logging	Configuration Guide
Message logging on IBM/CICS	IBM Release Notes
Sync API for IBM/CICS C programs	IBM Release Notes
Alpha/Open VMS release for OpenVMS 7.1	N/A



Programming Concepts

To use the API effectively, you need to understand how to use the following:

- Callback routines
- Item lists
- INI files
- Programming conventions
- Messaging limits

Before using the API functions, please review the information about these topics.



Callback Routines (NWDS_CALL_BACK)

A callback *routine*, sometimes referred to as a completion function, is a user-written function that executes when the I/O operation completes. To implement NetWeave's asynchronous function calls, you have to use callback routines.

A callback *structure* contains two elements:

- *Context*, a pointer to a persistent structure in the application's data space
- Callback function *address*, a pointer to the application's completion function

The application designer creates a callback structure (shown below) to contain all the information the application will need to continue its thread of operation when the NetWeave function call completes.

```
typedef struct {  
    NWDS_CONTEXT      context;  
    NWDS_CALL_BACK_PROC *procedure;  
} NWDS_CALL_BACK;
```

If the context points to something (i.e., is not NULL, which is a valid option), that something must be available when the callback function executes. Typically, it is not good programming practice to point to a stack variable, which may or may not be present when the callback is called. Although the context can point to a global or static data element, it should really point to a dynamically allocated memory buffer.

The second element of the callback structure is a pointer to the application's completion function. All completion functions must follow the format below:

```
typedef void (NWDS_CALL_BACK_PROC)(NWDS_CONTEXT, NWDS_ERRNO);
```



Parameter	Description
NWDS_CONTEXT	The same pointer that you passed to NetWeave as the first parameter of the callback structure. When the operation completes, NetWeave returns information to indicate the state of the application at the time the NetWeave call was initiated.
NWDS_ERRNO	The return code from the completion of the NetWeave operation.

An asynchronous function call is divided into two parts:

- The *initiation* phase, which is invoked when the user makes the NetWeave function call. If the operation returns NWDS_PENDING to the caller, it means that the operation is underway. If it returns an error, the operation never started. In some rare cases, the function invocation will return NWDS_SUCCESSFUL to indicate that the function has completed successfully and there will be no callback invocation.
- The *completion* phase, which NetWeave invokes after the remote operation has completed. To indicate the result of the remote operation, NetWeave passes an error condition to the completion routine.

To call the asynchronous library function calls synchronously, you can either pass a NULL directly, or set the pointer to the callback structure to NULL. In synchronous operation, when control returns to the application after calling a NetWeave function, all action is complete. In asynchronous operation, however, the return value of NWDS_PENDING indicates that although a message has been sent to the destination, the action is still in progress.



The Item List (NWDS_ITEM_LIST)

An item list is an array of elements of type NWDS_ITEM_LIST, as defined below:

```
typedef struct {  
    NWDS_ITEM_TYPE    type;  
    NWDS_SIZE         length;  
    Void              *item;  
}NWDS_ITEM_LIST;
```

A NetWeave *item list* lets you use system-specific features to change the usual action of a function call. You can use an item list to access special functions and features on supported remote platforms.

There are two types of item lists:

- *Control lists* use the values pointed to by the associated itemlist entry (i.e. the “item”) to change the function’s default operation.
- *Return lists* are used to retrieve information. If the call is asynchronous, the item must point to locations that are global, static, or dynamically allocated memory.

An item list ends with a standard element of type NWDS_END_OF_LIST. An empty (NULL) item list is an item list that contains only one element, of the type NWDS_END_OF_LIST.

Parameter	Description	Values
Item type	For each item in an itemlist, the type member indicates the type of item being supplied and specifies the optional parameters.	See netweave.h (the NetWeave header file).
Item length	The address stored in the item parameter points to the Item length value.	2 (type short) 4 (type long) The actual length of a control item list’s array of characters or The maximum length of data that can be copied to the address specified in a return item list’s buffer.
Item	Always holds an address	The value, of size item length, supplied to accompany the item type and length.



INI Files

NetWeave uses INI files on each node to configure a NetWeave application network for the following tasks:

Task	Description
Name translation	Translates logical names of network objects, such as processes, files, and tables, to physical names.
Data conversion	Describes the structure of network messages to enable NetWeave to convert data representations between different platforms.
Routing	Specifies how messages should travel from one platform to another through the NetWeave application network.
Performance parameters	Specifies parameters, such as buffer sizes, that can affect NetWeave performance. Typically, you will be using the default values for these parameters.

A NetWeave INI file uses the same syntax as the Microsoft Windows INI files. For more information about the INI files, see the *NetWeave Configuration Manual*.



Parameters and Pointers

The syntax example for each function shows the parameters that must be supplied for each function call. An asterisk (*) before an item name indicates that it is a pointer to the item, not the item itself. Most of the time NetWeave uses pointers. You can pass simple parameters, such as handles or sizes, directly as input when you make a function call. When these parameters are returned as output, NetWeave generally provides a pointer for the item.





Messaging Limits

Although message size cannot exceed 32567 bytes for both IPC and queued messaging, there is no limit to the number of fields in a message. For queued messaging, the amount of available disk storage space determines how many messages are allowed in a queue.





API Functions Grouped by Use

The table below lists the NetWeave API functional groups.

Used for	Function name	Description
Process initialization	nwds_batch	Starts a process on a remote system and waits for it to complete.
	nwds_execute	Starts a remote process and returns.
Initialization	nwds_init	Initializes the NWDS library.
	nwds_item_load_char	Handles itemlist loading functions.
	nwds_item_load_handle	
	nwds_item_load_long	
	nwds_item_load_short	
Identification	nwds_ping	Tests for existence of NetWeave connectivity.
	nwds_system_type	Determines type of remote system.
Timing/Sleep	nwds_exit	Returns NetWeave resources to system.
	nwds_stop	Terminates the process that nwds_execute started.
	nwds_sleep	Waits for event during asynchronous operation.
	nwds_sleep_callback	Callback function from nwds_sleep.
	nwds_sleep_clear_callback	Cancels nwds_sleep_callback.
	nwds_timer_start	Starts a NetWeave timer.
	nwds_timer_stop	Stops a NetWeave timer.
Translation	nwds_convert_data	Converts a record of a specified type from one platform's format to another's.
	nwds_convert_record	Translates a message into the format used by another system/compiler.

(continued)



Used for	Function name	Description
File management	nwds_file_close	Closes a file.
	nwds_file_copy	Copies a file.
	nwds_file_create	Creates a file.
	nwds_file_delete	Deletes a file.
	nwds_file_info	Retrieves information about a file.
	nwds_file_open	Opens a file.
	nwds_file_position	Completes a transaction involving a file/queue.
	nwds_file_read	Reads the first message from the file/queue.
	nwds_file_remove	Purges a file.
	nwds_file_update	Updates a specified record in a file.
	nwds_file_write	Appends a message to the end of the file /queue.
Naming	nwds_ini_delete_name	Deletes an INI file definition (memory only).
	nwds_ini_get_name	Retrieves an INI file definition.
	nwds_ini_put_name	Adds/modifies an INI file definition (memory only).
IPC	nwds_session_close	Returns system resources when terminating a multicasting session.
	nwds_ipc_accept	Accepts an IPC connection.
	nwds_ipc_broadcast	Sends a multicast message.
	nwds_ipc_connect	Tries to establish an IPC connection.
	nwds_ipc_options	Queries/modifies IPC parameters
	nwds_ipc_publish	Publishes the IPC name for subsequent connections.
	nwds_ipc_read	Reads an IPC message.
	nwds_ipc_register	Registers for multicast messages.
	nwds_ipc_shutdown	Terminates an IPC connection.
Security	nwds_ipc_write	Writes an IPC message.
	nwds_logoff	Terminates a secure session.
	nwds_logon	Establishes credentials for a secure session.
	nwds_msglog	Handles application level message logging to NetWeave logging facilities.
	nwds_password	Specifies a password for a secure session.

(continued)



Used for	Function name	Description
SQL	nwds_sql_column_bind	Handles SQL database operations.
	nwds_sql_column_count	
	nwds_sql_column_get	
	nwds_sql_column_info	
	nwds_sql_connect	
	nwds_sql_disconnect	
	nwds_sql_execute	
	nwds_sql_fetch	
	nwds_sql_select	
Transaction Processing	nwds_tp_abort	Aborts the transaction in progress.
	nwds_tp_commit	Commits the transaction in process.
	nwds_tp_resume	Resumes the transaction in process.
	nwds_tp_start	Starts a new transaction.
	nwds_tp_status	Gets the transaction status.
Notification	nwds_trigger_cancel	Cancels a trigger on a file.
	nwds_trigger_delete	Deletes a file trigger.
	nwds_trigger_read	Reads a file trigger.
	nwds_trigger_register	Registers for notification of file events.
Threads	nwds_dispatcher_create	Creates a thread-based connection dispatch service.
	nwds_dispatcher_stats	Retrieves the Dispatcher statistics (using the handle created by nwds_dispatcher_create).
	nwds_dispatcher_stop	Terminates Dispatcher operations.
Miscellaneous	nwds_error_text	Gets error text for the supplied error code.



Alphabetical Listing of Function Calls





NWDS_BATCH

This function, which is included in all NetWeave releases, starts a process on a remote system. The call completes when the remote process completes. If the process does not start, the `nwds_batch` function returns an error code that indicates why.

To execute a process on a remote system, you can use either `nwds_batch` (if the process will do something and then stop) or `nwds_execute` (if the process runs in the background, or needs to stay up indefinitely).

NWDS_ERRNO NWDS_BATCH

```
(char          *system_name,  
char          *cli_command,  
NWDS_ITEM_LIST *item_list,  
NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
system_name	✓		The system on which the batch process will run.
cli_command	✓		The command that the batch process will execute. Although the syntax is determined by the operating system on which the command will run, the command must consist of all printable characters and be terminated with a NULL byte.
item_list	✓		A pointer to an array of system-specific parameters. Any item list type appropriate for <code>nwds_execute</code> is usually appropriate for <code>nwds_batch</code> .
call_back	✓		A pointer to a callback structure that contains the function that NetWeave will call when <code>nwds_batch</code> completes. If <code>call_back</code> is set to NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_execute` on page 36.





NWDS_CONVERT_DATA

This function, which is included in all NetWeave releases, converts a record of a specified type from the format used by one computer to the format used by another. The `nwds_convert_data` call is synchronous and does not take an item list.

NOTE: Before calling `nwds_convert_data`, make sure the destination location has enough room to hold the converted data.

```
NWDS_ERRNO nwds_convert_data
(NWDS_DATA_CLASS      data_type,
 NWDS_SYSTEM_CLASS    to_system_type,
 NWDS_SYSTEM_CLASS    from_system_type,
 void                 *to_buffer,
 void                 *from_buffer);
```

Parameter	Input	Output	Description
<code>data_type</code>	✓		This value is either <code>NWDS_SHORT</code> or <code>NWDS_LONG</code> .
<code>to_system_type</code>	✓		The system type whose format the data will be converted to. Netweave.h lists all supported system types. Example: <code>NWDS_ERRNO nwds_convert_data(NWDS_LONG, NWDS_MSDOS, NWDS_UNIX_680xx, &dos_long, &unix_long);</code>
<code>from_system_type</code>	✓		The system type whose data you need to convert.
<code>to_buffer</code>		✓	The array where NetWeave places the converted value.
<code>from_buffer</code>	✓		An array in memory that contains the value to be converted.

Return Code (output)

Return code	Description
<code>NWDS_SUCCESSFUL</code>	The call completed successfully.
<code>NWDS_NOT_IMPLEMENTED</code>	The function is not available on the current platform.
<code>NWDS_NOT_INITIALIZED</code>	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
<code>NWDS_BAD_PARAMETER</code>	You are trying to call a function but one of your parameters is out of range.



For more information about the return codes, see page 229.

Related Functions

`nwds_convert_record` on page 23.







NWDS_CONVERT_RECORD

This function, which is included in all NetWeave releases, translates a message into the format used by another system/compiler. You can use `nwds_convert_record` for either of the following:

- Controlling all translation from the sending or receiving side
- Conversing with existing applications, such as Pathway serverclasses

Translation is controlled by the message name passed as the first argument. This name must match a group name in the INI file. Translation is also controlled by the alignment parameters of the source and target systems and compilers. The `nwds_convert_record` call is synchronous and does not take an item list or callback.

```
NWDS_ERRNO nwds_convert_record (  
    char          *message_name,  
    char          *source_alignment_parameters,  
    char          *target_alignment_parameters,  
    NWDS_SIZE     source_size,  
    void          *source,  
    NWDS_SIZE     maximum_target_size,  
    void          *target,  
    NWDS_SIZE     *return_size );
```

Parameter	Input	Output	Description
message_name	✓		The name of the INI file group that represents the DDL definition of the message in the application's INI file. For more information, see the <i>NetWeave Configuration Manual</i> .
source_alignment_parameters	✓		The name of the INI file group that defines the source system/compiler alignment parameters (see "Translation Rules" on the next page).
target_alignment_parameters	✓		The name of the INI file group that defines the destination system/compiler alignment parameters (see "Translation Rules" on the next page).
source_size	✓		The length in bytes of the source message in the source buffer.
source	✓		The address of the source buffer that contains the message that needs to be translated.
maximum_target_size	✓		The maximum length of the target buffer. NetWeave will not translate a message that exceeds the buffer size.
target	✓		The address of the target buffer that will contain the translated message.
return_size		✓	The actual length in bytes of the translated message.



Return Codes (output)

Return code	Description
NWDS_SUCCESSFUL	The call completed successfully.
NWDS_DATA_OVERFLOW	Translation terminated at the maximum target size. Increase the size of the destination buffer and try again.
NWDS_BAD_PARAMETER	The supplied DDL message name does not exist in the application's INI file. Locate the proper DDL definition and try again.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

Translation Rules

NetWeave uses alignment rules to map data types to physical storage for a particular compiler and/or system. The alignment rules also specify how the data types will be mapped in physical storage relative to each other.

For each system and compiler involved in message exchange, you must add an alignment (rules) group to your INI file. The rules group contains the following required parameters:

Parameter	Default	Description
SYSTEM_TYPE	None	The system identification value from the enum class in <code>netweave.h</code>
CHAR_ALIGNMENT	1	The number of bytes between the address boundaries of successive byte fields.
CHAR_SIZE	1	Size, in bytes of a character data element.
SHORT_ALIGNMENT	2	The number of bytes between the address boundaries of successive short integer fields.
SHORT_SIZE	2	The number of bytes to store an integer of type <code>SHORT</code> .
LONG_ALIGNMENT	4	The number of bytes between the address boundaries of successive long integer fields.
LONG_SIZE	4	The number of bytes to store an integer of type <code>LONG</code> .

The example below shows how to configure your INI file to use these special translation features. Assume that the source system is a PC with 32-bit architecture and compiler, and the destination system is a legacy service application on a remote platform (a Tandem host). The PC sends a request message to the host and receives a reply message.

NOTE: Because the PC aligns short integers on 32-bit boundaries; we override the default short alignment rule for the PC. (The alignment rule offsets the second field in the reply to the fourth byte position for the PC, and to the second byte position for the host.)



```
[REQUEST_MESSAGE]
```

```
DDL_ENTRY = 1
```

```
DDL_FIELD_COUNT = 3
```

```
DDL_FIELD_1 = CHAR 3
```

```
DDL_FIELD_2 = SHORT 2
```

```
DDL_FIELD_3 = LONG 4
```

```
[REPLY_MESSAGE]
```

```
DDL_ENTRY = 1
```

```
DDL_FIELD_COUNT = 2
```

```
DDL_FIELD_1 = CHAR 1
```

```
DDL_FIELD_2 = SHORT 2
```

```
[PC_RULES]
```

```
SYSTEM_TYPE = NWDS_MS_WIN32
```

```
SHORT_ALIGNMENT = 4
```

```
[HOST_RULES]
```

```
SYSTEM_TYPE = NWDS_NONSTOP_WIDE
```

Related Functions

`nwds_convert_data` on page 20.





NWDS_DISPATCHER_CREATE

This function creates a Dispatcher and returns a handle that may be used on subsequent Dispatcher operations such as `nwds_dispatcher_stats` and `nwds_dispatcher_stop`. The `nwds_dispatcher_create` function is included in all NetWeave releases. The call is synchronous and does not take an item list or callback.

The process of creating a Dispatcher also creates an internal NetWeave thread known as the *boss thread*. After creating the boss thread, NetWeave creates additional application threads called “worker” threads. The Dispatcher threads process messages from remote client applications. When a new client attaches to the Dispatcher thread, a new worker thread is created to service this client. The worker thread procedure has one parameter (`applcontext`) that stores the program’s context.

The boss thread acts as the intermediary between the external client and the worker threads until one of them terminates its connection with the boss. The boss then terminates the connection to the other.

```
typedef void      (NWDS_APPLTHREAD_PROC) (void *);  
  
NWDS_ERRNO nwds_dispatcher_create  
    (char          *ext_pub_name,  
     char          *int_pub_name,  
     char          *ctrl_pub_name,  
     void          *applcontext,  
     NWDS_HANDLE   *dispatcher_handle,  
     NWDS_APPLTHREAD_PROC worker_thread);
```

Parameter	Input	Output	Description
ext_pub_name	✓		The publish name to which remote client applications connect. The publish name (a NULL-terminated string) is a protocol group in the application's INI file.
int_pub_name	✓		The internal publish name that a worker thread uses to communicate with the boss thread. This logical name (a NULL-terminated string) is a protocol group in the application's INI file.
ctrl_pub_name	✓		The publish name that an external program uses to send management requests to the boss thread. The boss thread can return statistics and be instructed to shut down gracefully. The public name (a NULL-terminated string) is a protocol group in the application's INI file.
applcontext	✓		The address of a context area that is passed to each worker thread upon startup.
dispatcher_handle		✓	The address where the handle for the control connection will be stored. <code>nwds_dispatcher_stats</code> and <code>nwds_dispatcher_stop</code> use this handle as a parameter.
worker_thread	✓		A pointer to the worker thread's processing routine of type <code>NWDS_APPLTHREAD_PROC</code> .



Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	The call completed successfully.
NWDS_BAD_PARAMETER	You are trying to call a function but one of your parameters is out of range.
NWDS_NO_MEMORY	Could not allocate enough memory to pass parameters to the boss thread.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_NOTHREAD	Could not create a boss thread. Platform is not threaded.
NWDS_OPERATION_FAILED	The call did not complete successfully.

Related Functions

`nwds_dispatcher_stats` on page 30.

`nwds_dispatcher_stop` on page 32.





NWDS_DISPATCHER_STATS

This function, which is included in all NetWeave releases, retrieves the following statistics from the Dispatcher:

- The number of active worker threads
- The number of messages transmitted and received since startup

The `nwds_dispatcher_stats` call is synchronous and does not take an item list.

```
NWDS_ERRNO nwds_dispatcher_stats
(
    NWDS_HANDLE dispatcher_handle,
    int          *num_threads,
    int          *num_messages);
```

Parameter	Input	Output	Description
dispatcher_handle	✓		The handle returned from <code>nwds_dispatcher_create</code> .
num_threads		✓	The address of an integer that will receive the number of active worker threads associated with the <code>dispatcher_handle</code> .
num_messages		✓	The address of an integer that will receive the number of messages that have passed through the boss thread.

Return Code (output)

Return code	Description
NWDS_BAD_PARAMETER	NULL pointers were passed as arguments.
NWDS_NO_MEMORY	Could not allocate enough memory to pass parameters to the boss thread.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOTHREAD	Could not create a boss thread. Platform is not threaded.
NWDS_OPERATION_FAILED	The call did not complete successfully.
NWDS_SUCCESSFUL	The call completed successfully.

Related Functions

`nwds_dispatcher_create` on page 27.

`nwds_dispatcher_stop` on page 32.





NWDS_DISPATCHER_STOP

This function, which is included in all NetWeave releases, terminates a Dispatcher created by `nwds_dispatcher_create`. The `nwds_dispatcher_stop` call is synchronous and does not take an item list.

```
NWDS_ERRNO nwds_dispatcher_stop
    (NWDS_HANDLE      dispatcher_handle);
```

Parameter	Input	Output	Description
dispatcher_handle	✓		The handle that <code>nwds_dispatcher_create</code> returns.

Return Code (output)

Return code	Description
NWDS_BAD_PARAMETER	NULL pointers were passed as arguments.
NWDS_NO_MEMORY	Could not allocate enough memory to pass parameters to the boss thread.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_NOTHREAD	Could not create a boss thread. Platform is not threaded.
NWDS_OPERATION_FAILED	The call did not complete successfully.
NWDS_SUCCESSFUL	The call completed successfully.

Related Functions

`nwds_dispatcher_create` on page 27.

`nwds_dispatcher_stats` on page 30.





NWDS_ERROR_TEXT

This function, which is included in all NetWeave releases, converts a NetWeave status code into a text string. The `nwds_error_text` call is synchronous and does not take an item list.

```
NWDS_ERRNO nwds_error_text
    (NWDS_ERRNO    status_code,
     char          *text_string);
```

Parameter	Input	Output	Description
status_code	✓		A value returned from a NetWeave function call.
text_string		✓	A line of text (a NULL-terminated string) that tells users what the status code means.

Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	The call completed successfully.
NWDS_BAD_PARAMETER	Status code represents a non-NetWeave error code.

Related Functions

`nwds_msglog` on page 126.





NWDS_EXECUTE

This function starts a process on a designated system, and completes when the process starts. If the process starts successfully, use the `return_list` item list to find out more about it. If the process fails to start, the return code tells you why. The `nwds_execute` function is included in all NetWeave releases.

NOTE: To stop a process started by `nwds_execute`, use `nwds_stop`.

To execute a process on a remote system, you can use either `nwds_batch` or `nwds_execute`. If the process will do something and then stop, use `nwds_batch`. If the process runs in the background, or needs to continue running indefinitely, use `nwds_execute`.

```
NWDS_ERRNO nwds_execute
    (char          *image_name,
     NWDS_ITEM_LIST *control_list,
     NWDS_ITEM_LIST *return_list,
     NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
image_name	✓		The name (a NULL-terminated string) of the executable file you want to run, either a logical name or a physical name expressed in the syntax of the target system. If it's a logical name, a group in the process INI file must contain a translation to a physical name. The first part of the physical name may be the NetWeave node name of the target system where the file is located. If there is no node prefix, NetWeave looks for the file on the local system.
control_list	✓		A pointer to an array of system-specific runtime parameters. (See "The Control_list Parameters" on the next page.)
return_list		✓	A pointer to an array of system-specific parameters that describe the program (job). For more information, see "Using Return_list to Stop a Remote Process" on page 38.
call_back	✓		A pointer to a callback structure that contains the function that will be called when <code>nwds_execute</code> completes. If NULL, the call is synchronous.



Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling nwds_init() and before calling nwds_exit(). Call nwds_init() and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

The Control_List Parameters

(For CICS only) To write to a queue, use NWDS_CICS_QUEUE and NWDS_CICS_REQ_DATA. Use the other NWDS_CICS parameters with EXEC CICS START.

The table below lists the system-specific control_list runtime parameters:

Platform	Parameter	Description
MVS/CICS	NWDS_CICS_IMAGE_FLAG	A long; indicates whether you are writing to TDQ or using CICS START.
	NWDS_CICS_REQID	A string; REQID field value.
	NWDS_CICS_REQ_DATA	An array of bytes; the data to pass to the task.
	NWDS_CICS_REQ_FLAG	A long; flag if data has FMH format.
	NWDS_CICS_TIME_FLAG	A long; flag AFTER or AT time.
	NWDS_CICS_HOURS	A long; hours value.
	NWDS_CICS_MINUTES	A long; minutes value.
	NWDS_CICS_SECONDS	A long; seconds value.
	NWDS_CICS_RTRANSID	A string; RTRANSID to pass to started task.
	NWDS_CICS_RTERMID	A string; RTERMID to pass to started task.
	NWDS_CICS_QUEUE	A string; queue name to pass to started task.
DEC	NWDS_VMS_DELAY_TIME	A long; the number of seconds to wait after sys\$creprc finishes testing that the process is still active.
	NWDS_VMS_DETACHED	A long; if it is non-zero, run the job in the background.
	NWDS_VMS_PRIORITY	An integer; the priority at which the job runs.
	NWDS_VMS_PROCESS_NAME	A byte array; the name of the process.
	NWDS_VMS_WAIT_ATTEMPTS	Not currently used.

(continued)



Platform	Parameter	Description
Tandem	NWDS_TAN_ASSIGNMSG	A byte array; you may define more than one Guardian assign message.
	NWDS_TAN_HOMETERM	A byte array; the home terminal of the process is the default for stdin, stdout, and stderr.
	NWDS_TAN_MEMORYPAGE	An integer; the initial allocation of memory.
	NWDS_TAN_PARAMMSG	A byte array; a Guardian param message (you may define more than one).
	NWDS_TAN_PRIORITY	An integer; the priority assigned to the process.
	NWDS_TAN_PROCESSNAME	A string; this is returned in the return_list, and may be passed to nwds_stop.
	NWDS_TAN_PROCESSOR	An integer; the CPU in which the process will start.
	NWDS_TAN_STARTUPMSG	A byte array; the process' startup message.
UNIX	NWDS_UNIX_COMMAND_ARGS	A string; the runtime command arguments.
	NWDS_UNIX_RETURN_PID	A string; this is returned in the return_list and may be passed to nwds_stop.
WIN32	NWDS_WIN32_COMMAND_ARGS	A string; the runtime command arguments.
	NWDS_WIN32_RETURN_PID	A string; this is returned in the return_list and may be passed to nwds_stop.

Using Return_list to Stop a Remote Process

To stop a remote process, whether started with `nwds_execute` or not, you must include the following parameter(s) in the `return_list`:

Platform	Parameter	Description
DEC	NWDS_VMS_RETURN_NAME	A byte array; the name you passed as <code>VMS_PROCESS_NAME</code> .
	NWDS_VMS_RETURN_PID	A long integer; the VMS process ID.
Tandem	NWDS_TAN_PROCESSNAME	A string, in the format for passing to <code>nwds_stop</code> .
UNIX	NWDS_UNIX_RETURN_PID	A string, in the format for passing to <code>nwds_stop</code> .
WIN32	NWDS_WIN32_RETURN_PID	A string, in the format for passing to <code>nwds_stop</code> .



NOTE: The underlying operating system call determines how `nwds_execute` works. On some systems, this call may return true even if the process did not really start. For example, on an VMS system, if the process has quota or privilege problems, `nwds_execute` will return `NWDS_SUCCESSFUL` even if the process did not start successfully.

Related Functions

`nwds_batch` on page 18







NWDS_EXIT

This function, which is included in all NetWeave releases, shuts down any open connections and releases all system resources. On some systems (such as Windows running TCP/IP), if you do not call `nwds_exit`, NetWeave can't tell the TCP/IP device driver to free any socket handles that NetWeave may have allocated.

NOTE: Before calling `nwds_init` again for any reason, you must first call `nwds_exit`. If you don't, you will have memory leaks within the NetWeave application.

```
NWDS_ERRNO nwds_exit (void);
```

Return Code (output)

If `nwds_exit` returns a value other than `NWDS_SUCCESSFUL`, there was an unrecoverable system error.

Related Functions

`nwds_sleep` on page 136.

`nwds_sleep_callback` on page 138.

`nwds_sleep_clear_callback` on page 141.

`nwds_stop` on page 164.

`nwds_timer_start` on page 169.

`nwds_timer_stop` on page 172.





NWDS_FILE_CLOSE

This function closes a file opened by an earlier call to `nwds_file_open`. The `nwds_file_close` function is provided as part of NetWeave's Data Server and/or Message Queueing options.

NOTE: Any files that `nwds_file_open` opens will be closed automatically when the application terminates, or if you call `nwds_exit`.

```
NWDS_ERRNO nwds_file_close
    (NWDS_HANDLE          file_handle,
     NWDS_ITEM_LIST       *control_items,
     NWDS_CALL_BACK       *call_back);
```

Parameter	Input	Output	Description
file_handle	✓		The handle returned from a call to <code>nwds_file_open</code> .
control_items	✓		Because <code>nwds_file_close</code> does not use control items, <code>control_items</code> should be set to NULL.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	The call completed successfully.
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_NO_MEMORY	The system is overloaded; process out of memory.
NWDS_PENDING	The operation has been initiated successfully. Final status and data will be delivered to the specified callback function.

For more information about the return codes, see page 229.

Related Functions

`nwds_file_copy` on page 46.

`nwds_file_create` on page 48.

`nwds_file_delete` on page 52.



`nwds_file_info` on page 55.

`nwds_file_open` on page 59.

`nwds_file_position` on page 63.

`nwds_file_read` on page 67.

`nwds_file_remove` on page 70.

`nwds_file_update` on page 73.

`nwds_file_write` on page 76.







NWDS_FILE_COPY

The `nwds_file_copy` function copies a file from one system to another. This function is provided as part of NetWeave's File Transfer option.

You may use `nwds_file_copy` to transfer both text files and binary images. If the INI file contains DDL entries for the file, the data may be converted. Use record blocking to optimize transfer through the communications layer. Text files are created on the destination system in a format compatible with that system's standard text editor (`nwds_file_copy` does not overwrite non-empty destination files).

```
NWDS_ERRNO nwds_file_copy
(
    char          *source_file_name,
    NWDS_ITEM_LIST *source_items,
    char          *destination_file_name,
    NWDS_ITEM_LIST *destination_items,
    NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
source_file_name	✓		The name (a NULL-terminated string) of the file you want to copy. The name can be either a logical name, or a physical file name expressed in the syntax of the target system. If it's a logical name, a group in the process INI file must contain a translation to a physical name. The first part of the physical name may be the NetWeave node name of the target system from which the file will be copied. If there is no node prefix, NetWeave looks for the file on the local system.
source_items	✓		A pointer to an array of system-specific parameters. See "Considerations" on the next page.
destination_file_name	✓		The name (a NULL-terminated string) of the new file, either a logical name or a physical file name expressed in the syntax of the target system. If it's a logical name, a group in the process INI file must contain a translation to a physical name. The first part of the physical name may be a NetWeave node name of the system to which the file will be copied. If there is no node prefix, NetWeave looks for the file on the local system. If the destination file does not exist, NetWeave creates the destination file.
destination_items	✓		A pointer to a system-specific parameter. See "Considerations" on the next page.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.



Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	The call completed successfully.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_NO_MEMORY	The system is overloaded; process out of memory.
NWDS_PENDING	The operation has been initiated successfully. Final status and data will be delivered to the specified callback function.

For more information about the return codes, see page 229.

Example

Windows copying a file from Tandem to Alpha/VMS:

```
nwds_file_copy (  
    "TANDEM::fileA",  
    source_items,  
    "VAX::fileB",  
    target_items,  
    NULL);
```

Considerations

The `source_items` parameters specify the source platform's file format (`NWDS_FILE_BLOCKING`) and a valid file type (`NWDS_FILE_TYPE`).

Source_items parameter	Value	Description
NWDS_FILE_BLOCKING	NWDS_FILE_BLOCKING_ON	NetWeave blocks logical records into physical records.
	NWDS_FILE_BLOCKING_OFF	No blocking is done.
NWDS_FILE_TYPE	NWDS_FILE_TYPE_C	C file
	NWDS_FILE_TYPE_FIFO	FIFO file
	NWDS_FILE_TYPE_LEGACY	Legacy file (system-dependent)



The `NWDS_FILE_TYPE` value in turn determines which items are returned, as shown below:

Type	NetWeave definition	Description
C Files	NWDS_CFILE_SHARING	A logical value that defines the access mode: NWDS_CFILE_READ_WRITE: read and write access NWDS_CFILE_READ_ONLY: read-only access NWDS_CFILE_WRITE_ONLY: write-only access NWDS_CFILE_APPEND: write to the end of the file
	NWDS_CFILE_FORMAT	A logical value that characterizes the file: NWDS_CFILE_TEXT: newline delineated text not necessarily printable; uses <code>fgets</code> , <code>fputs</code> NWDS_CFILE_BINARY: no format; uses <code>fwrite</code> , <code>fread</code> NWDS_CFILE_VARIABLE: NetWeave special file type; each record consists of a 2-byte length followed by data
FIFO	NWDS_FIFO_SHARING	A logical that defines what to do after a message is read: NWDS_FIFO_READ_HOLD_POS: the current pointer is not updated after a read NWDS_FIFO_READ_NEW_POS: the current pointer is changed after a read NWDS_FIFO_APPEND_ONLY: messages are added to the end of the file
Legacy	If you need to copy legacy files between proprietary file systems, see <code>netweave.h</code> for system-specific features.	

For the destination system in the copy operation, `destination_items` is analogous to `source_items`.

Related Functions

`nwds_file_close` on page 43.

`nwds_file_create` on page 48.

`nwds_file_delete` on page 52.

`nwds_file_info` on page 55.

`nwds_file_open` on page 59.

`nwds_file_position` on page 63.

`nwds_file_read` on page 67.

`nwds_file_remove` on page 70.

`nwds_file_update` on page 73.

`nwds_file_write` on page 76.



NWDS_FILE_CREATE

The `nwds_file_create` function creates a file. To define the new file's characteristics, you need a properly formatted item list. With an appropriate item list, `nwds_file_create` can construct any file type on the target system. The `nwds_file_create` function is provided as part of NetWeave's Data Server and/or Message Queueing options.

```
NWDS_ERRNO nwds_file_create
    (char                *file_name,
     NWDS_ITEM_LIST      *control_items,
     NWDS_CALL_BACK      *call_back);
```

Parameter	Input	Output	Description
file_name	✓		The name (a NULL-terminated string) of the file you want to create. The name can be either a logical name, or a physical file name expressed in the syntax of the target system. If it's a logical name, a group in the process INI file must contain a translation to a physical name. The first part of a physical name may be the NetWeave node name of the target system where the file will be created. If there is no node prefix, NetWeave will try to create the file on the local system.
control_items	✓		A pointer to system-specific parameters that specify the file type and its characteristics. See "Considerations" below.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.

For more information about the return codes, see page 229.



Considerations

The `control_items` parameter contains one or more of the following system-specific parameters that indicate a valid file type, `NWDS_FILE_TYPE`, for the destination platform:

- `NWDS_FILE_TYPE_C` (C file)
- `NWDS_FILE_TYPE_FIFO` (FIFO file)
- `NWDS_FILE_TYPE_LEGACY` (Legacy file, system-dependent)

Type	NetWeave definition	Description
C file	<code>NWDS_CFILE_EOF</code>	A long; the number of bytes in the file.
FIFO	<code>NWDS_FIFO_MAX_SEGMENTS</code>	An integer; the number of segments in the file.
	<code>NWDS_FIFO_SEGMENT_SIZE</code>	An integer; the number of bytes in a segment.
Legacy	For system-specific features when creating a legacy file in a proprietary file system, please see <code>netweave.h</code> .	

Related Functions

`nwds_file_close` on page 43.

`nwds_file_copy` on page 46.

`nwds_file_delete` on page 52.

`nwds_file_info` on page 55.

`nwds_file_open` on page 59.

`nwds_file_position` on page 63.

`nwds_file_read` on page 67.

`nwds_file_remove` on page 70.

`nwds_file_update` on page 73.

`nwds_file_write` on page 76.





NWDS_FILE_DELETE

This function deletes the record or row entry at the current file position. The `nwds_file_delete` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_file_delete
    (NWDS_HANDLE          file_handle,
     NWDS_ITEM_LIST       *control_items,
     NWDS_CALL_BACK       *call_back);
```

Parameter	Input	Output	Description
file_handle	✓		The handle returned from a call to <code>nwds_file_open</code> .
control_items	✓		A pointer to an array of system-specific parameters. To ensure transaction protection for Tandem, use <code>NWDS_TP_HANDLE</code> with a TP handle that was returned from a call to <code>NWDS_TP_START</code> .
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.

For more information about the return codes, see page 229.

Related Functions

`nwds_file_close` on page 43.

`nwds_file_copy` on page 46.

`nwds_file_create` on page 48.

`nwds_file_info` on page 55.

`nwds_file_open` on page 59.

`nwds_file_position` on page 63.



`nwds_file_read` on page 67.

`nwds_file_remove` on page 70.

`nwds_file_update` on page 73.

`nwds_file_write` on page 76.







NWDS_FILE_INFO

This function retrieves the information you requested about a particular file. The file does not have to be open. The `nwds_file_info` function is provided as part of NetWeave's Data Server and/or Message Queue options.

Before the call, to specify which information you need from the file, place the types of the requested items in the `return_items` list structure. If the call is asynchronous, use persistent memory to receive the values of the requested information. Do not declare return items on the local stack. Place the addresses of these locations for the returned information in the item list. When the function call completes, NetWeave copies the values of the items to the specified locations.

```
NWDS_ERRNO nwds_file_info
    (char          *file_name,
     NWDS_ITEM_LIST *control_items,
     NWDS_ITEM_LIST *return_items,
     NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
file_name	✓		The name (a NULL-terminated string) of the file whose information you want to retrieve. The name can be either a logical name, or a physical file name expressed in the syntax of the target system. If it's a logical name, a group in the process INI file must contain a translation to a physical name. The first part of a physical name may be the NetWeave node name of the target system where the file is located. If there is no node prefix, NetWeave looks for the file on the local system.
control_items	✓		A pointer to an array of system-specific parameters that modify the default operation of this function. For more information, see "Considerations" on the next page.
return_items		✓	A pointer to an array of system-specific parameters. See "Considerations" on the next page.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.



Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.

For more information about the return codes, see page 229.

Considerations

For `control_items`, the `NWDS_FILE_TYPE` determines which items are returned:

- `NWDS_FILE_TYPE_C` (C file)
- `NWDS_FILE_TYPE_FIFO` (FIFO file)
- `NWDS_FILE_TYPE_LEGACY` (Legacy file, system-dependent)

Each file type can return the following:

File type	NetWeave definition	Description
C file	<code>NWDS_CFILE_EOF</code>	The number of bytes in the file.
FIFO	<code>NWDS_FIFO_DATA_SIZE</code>	Number of bytes in a segment (the same as the value <code>NWDS_FIFO_SEGMENT_SIZE</code> provided in <code>NWDS_FILE_CREATE</code>).
	<code>NWDS_FIFO_SEGMENT_COUNT</code>	Number of segments in use in a queue.
	<code>NWDS_FIFO_NUMBER_RECORDS</code>	Number of logical records (not segments) in the queue.
	<code>NWDS_FIFO_FREE_SEGMENTS</code>	Number of free segments in a queue.
Legacy	For system-specific features when querying legacy files in a proprietary file system, please see <code>netweave.h</code> .	

Related Functions

`nwds_file_close` on page 43.

`nwds_file_copy` on page 46.

`nwds_file_create` on page 48.

`nwds_file_delete` on page 52.

`nwds_file_open` on page 59.



`nwds_file_position` on page 63.

`nwds_file_read` on page 67.

`nwds_file_remove` on page 70.

`nwds_file_update` on page 73.

`nwds_file_write` on page 76.







NWDS_FILE_OPEN

This function opens a file using its logical or physical name. The file type and the system on which the file resides determine how a file is opened. To set any other file open conditions, use an item list with item types and values appropriate for the target file system.

The `nwds_file_open` function is provided as part of NetWeave's Data Server and/or Message Queue options.

```
NWDS_ERRNO nwds_file_open
    (char          *file_name,
     NWDS_HANDLE   *file_handle,
     NWDS_ITEM_LIST *control_items,
     NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
file_name	✓		The name (a NULL-terminated string) of the file you want to open. The name may be either a logical name, or a physical file name expressed in the syntax of the target system. If it's a logical name, a group in the process INI file must contain a translation to a physical name. The first part of a physical name may be the NetWeave node name of the target system where the file is located. If there is no node prefix, NetWeave will look for the file on the local system.
file_handle		✓	The identifier to use for subsequent operations on this file.
control_items	✓		A pointer to an array of system-specific parameters. For more information, see "Considerations" on the next page.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_PENDING	The operation was initiated successfully. Final status and data will be delivered to the specified callback function.

For more information about the return codes, see page 229.



Considerations

The `control_items` parameter points to the parameters that specify the source platform's file format (NWDS_FILE_BLOCKING) and valid file type (NWDS_FILE_TYPE).

control_items parameter	Value	Description
NWDS_FILE_BLOCKING	NWDS_FILE_BLOCKING_ON	NetWeave blocks logical records into physical records.
	NWDS_FILE_BLOCKING_OFF	No blocking is done.
NWDS_FILE_TYPE	NWDS_FILE_TYPE_C	C file
	NWDS_FILE_TYPE_FIFO	FIFO file
	NWDS_FILE_TYPE_LEGACY	Legacy file (system-dependent)

The NWDS_FILE_TYPE value determines which items are returned, as shown below:

Type	NetWeave definition	Description
C Files	NWDS_CFILE_SHARING	A logical; defines the access mode: NWDS_CFILE_READ_WRITE: read and write access NWDS_CFILE_READ_ONLY: read-only access NWDS_CFILE_WRITE_ONLY: write-only access NWDS_CFILE_APPEND: write to the end of the file
	NWDS_CFILE_FORMAT	A logical; characterizes the file: NWDS_CFILE_TEXT: newline delineated text, not necessarily printable; uses fgets, fputs NWDS_CFILE_BINARY: no format; uses fwrite, fread NWDS_CFILE_VARIABLE: a NetWeave special file type, where each record consists of a 2-byte length followed by the data
FIFO	NWDS_FIFO_SHARING	A logical; defines what to do after a message is read: NWDS_FIFO_READ_HOLD_POS: the current pointer is not updated after a read. NWDS_FIFO_READ_NEW_POS: the current pointer is changed after a read. NWDS_FIFO_APPEND_ONLY: messages are added to the end of the file.
Legacy	For system-specific features when querying legacy files in a proprietary file system, please see netweave.h.	



Related Functions

`nwds_file_close` on page 43.

`nwds_file_copy` on page 46.

`nwds_file_create` on page 48.

`nwds_file_delete` on page 52.

`nwds_file_info` on page 55.

`nwds_file_position` on page 63.

`nwds_file_read` on page 67.

`nwds_file_remove` on page 70.

`nwds_file_update` on page 73.

`nwds_file_write` on page 76.





NWDS_FILE_POSITION

This function sets or resets the record position in the file. The `nwds_file_position` function is provided as part of NetWeave's Data Server and/or Message Queue options.

The parameters for setting the pointer are passed in the *control* item list structure. For certain types of files, `nwds_file_position` can return information about the current record pointer or file position. Before making the call, place the types of the items for which you are requesting information in the *return* item list structure. When the call returns, the values of the items are copied to the designated item locations. For more information, see `nwds_file_info` on page 55.

For message queues, if you call `nwds_file_position` with an empty item list, it advances the head pointer of a message queue. For more information about transaction processing with message queues, see "Message Queue Files" on page 193.

```
NWDS_ERRNO nwds_file_position
    (NWDS_HANDLE          file_handle,
     NWDS_ITEM_LIST       *control_items,
     NWDS_ITEM_LIST       *return_items,
     NWDS_CALL_BACK       *call_back);
```

Parameter	Input	Output	Description
file_handle	✓		The identifier returned from a call to <code>nwds_file_open</code> .
control_items	✓		A pointer to an array of system-specific parameters. For more information, see "Considerations" below.
return_items		✓	A pointer to an array of system-specific parameters. Include in the return item list the types for which you want values to be returned, and the addresses of locations to which values will be copied. See "Considerations" below.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_PENDING	The operation has been initiated successfully. Final status and data will be delivered to the specified callback function.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

For more information about the return codes, see page 229.



Considerations

For files that don't have keys and indices, addressing can be done in either of two ways:

- For unstructured files (no concept of a record), use a relative byte address.
- For structured files, positioning is by relative record number. File position is system- and filetype-specific.

For file structures with keys and indices, item types supported on the host platform can define an individual record or the first of a sequential set of records. Although item types differ for each host, all legacy file systems that support indexed access use an item type to specify each of the following:

- Which index to use
- Which key value to compare
- How to compare the key with the keys in the record set

The `control_items` parameter gives NetWeave additional information about the exact positioning requirements for the file system being accessed.

File type	Name	Description
C Files	NWDS_CFILE_FTELL	A long; the current relative byte address in a CFILE flat file.
	NWDS_CFILE_SEEK_OFFSET	A long; the new byte position relative to the starting point specified by SEEK_TYPE.
	NWDS_CFILE_SEEK_TYPE	Tells which starting point to use.
	NWDS_CFILE_SEEK_SET	Start from the beginning of the file.
	NWDS_CFILE_SEEK_CURRENT	Start from the current byte position.
	NWDS_CFILE_SEEK_END	Start relative to the end of the file.
FIFO	NWDS_FIFO_SEEK_OFFSET	An integer; the number of messages to advance or retreat. The default value is -1, as in "take 1 off". Use a positive value to reset the head pointer to reread previous messages (put it back on the queue). Use a negative value to remove messages from the queue.
Legacy	For system-specific features when querying legacy files in a proprietary file system, please see <code>netweave.h</code> .	

Related Functions

`nwds_file_close` on page 43.

`nwds_file_copy` on page 46.

`nwds_file_create` on page 48.

`nwds_file_delete` on page 52.



`nwds_file_info` on page 55.

`nwds_file_open` on page 59.

`nwds_file_read` on page 67.

`nwds_file_remove` on page 70.

`nwds_file_update` on page 73.

`nwds_file_write` on page 76.







NWDS_FILE_READ

This function retrieves the record or row at the current file pointer. The `nwds_file_read` function is provided as part of NetWeave's Data Server and/or Message Queue options.

Where supported, record locking is indicated in an item type passed in the item list structure. If an empty item list is passed, the record is not locked. Depending on the properties of the underlying file system, NetWeave may return the argument to the callback for asynchronous invocations of `nwds_file_read`. Some hosts support read-only access that bypasses another user's lock. To lock or unlock a record, please see `netweave.h` for specific information for each platform and type of file structure.

```
NWDS_ERRNO nwds_file_read
(
    NWDS_HANDLE      file_handle,
    NWDS_SIZE         buffer_size,
    void             *buffer,
    NWDS_SIZE         *return_size,
    NWDS_ITEM_LIST    *control_items,
    NWDS_ITEM_LIST    *return_items,
    NWDS_CALL_BACK    *call_back);
```

Parameter	Input	Output	Description
file_handle	✓		The identifier returned from a call to <code>nwds_file_open</code> .
buffer_size	✓		The maximum number of bytes that may be copied to the buffer (next parameter). For a record-oriented file, one record is read from the file. If it is bigger than <code>buffer_size</code> , <code>NWDS_DATA_OVERFLOW</code> is returned and no data is transferred. For a flat file, up to <code>buffer_size</code> bytes are read from the file.
buffer	✓		The address of the array where the record is returned.
return_size		✓	The actual number of bytes copied to the buffer.
control_items	✓		A pointer to an array of system-specific parameters. When positioning to records in files in proprietary file systems, see <code>netweave.h</code> for system-specific features that affect how the <code>control_items</code> parameter is used.
return_items		✓	A pointer to an array of system-specific parameters. In the return item list, include the types for which values will be returned and the addresses of locations to which values will be copied.
call_back	✓		A pointer to the callback structure. If <code>NULL</code> , the call is synchronous.



Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

If another user locks the record, the error `NWDS_RECORD_IS_LOCKED` is returned. NetWeave does not wait for the record to become unlocked. For more information about the return codes, see page 229.

Related Functions

`nwds_file_close` on page 43.

`nwds_file_copy` on page 46.

`nwds_file_create` on page 48.

`nwds_file_delete` on page 52.

`nwds_file_info` on page 55.

`nwds_file_open` on page 59.

`nwds_file_position` on page 63.

`nwds_file_remove` on page 70.

`nwds_file_update` on page 73.

`nwds_file_write` on page 76.





NWDS_FILE_REMOVE

This function deletes a remote file. It fails if any process has the file open. The `nwds_file_remove` function is provided as part of NetWeave's Data Server and/or Message Queue options.

```
NWDS_ERRNO nwds_file_remove
(char          *file_name,
NWDS_ITEM_LIST *control_items,
NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
file_name	✓		The name (a NULL-terminated string) of the file to remove. The name may be either a logical name, or a physical file name expressed in the syntax of the target system. If it's a logical name, a group in the process INI file must contain a translation to a physical name. The first part of a physical name may be a NetWeave node name that indicates the target system where the file is located. If there is no node prefix, NetWeave will look for the file on the local system.
control_items	✓		A pointer to an array of system-specific parameters. Use <code>control_items</code> to specify the type of file (NWDS_FILE_TYPE) you intend to remove: <ul style="list-style-type: none">• NWDS_FILE_TYPE_C (C file)• NWDS_FILE_TYPE_FIFO (FIFO file)• NWDS_FILE_TYPE_LEGACY (Legacy file, system-dependent)
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.

For more information about the return codes, see page 229.



Related Functions

`nwds_file_close` on page 43.

`nwds_file_copy` on page 46.

`nwds_file_create` on page 48.

`nwds_file_delete` on page 52.

`nwds_file_info` on page 55.

`nwds_file_open` on page 59.

`nwds_file_position` on page 63.

`nwds_file_read` on page 67.

`nwds_file_update` on page 73.

`nwds_file_write` on page 76.







NWDS_FILE_UPDATE

This function changes the data fields in an existing record or row. The `nwds_file_update` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_file_update
    (NWDS_HANDLE      file_handle,
     NWDS_SIZE        buffer_size,
     void             *buffer,
     NWDS_ITEM_LIST   *control_items,
     NWDS_CALL_BACK   *call_back);
```

Parameter	Input	Output	Description
file_handle	✓		The handle returned from a call to <code>nwds_file_open</code> .
buffer_size	✓		The number of bytes to be copied from the buffer (next parameter in this table, below). For a record-oriented file, if you try to exceed the maximum record size specified for the file, NetWeave returns <code>NWDS_DATA_OVERFLOW</code> . For a flat file, <code>buffer_size</code> bytes are replaced in the file.
buffer	✓		The address of the array that contains the changed data.
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to the callback structure. If <code>NULL</code> , the call is synchronous.

Return Code (output)

Return code	Description
<code>NWDS_NOT_IMPLEMENTED</code>	The function is not available on the current platform.
<code>NWDS_NOT_INITIALIZED</code>	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
<code>NWDS_BAD_HANDLE</code>	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

If another user locks the record, the error `NWDS_RECORD_IS_LOCKED` is returned. NetWeave does not wait for the record to become unlocked. For more information about the return codes, see page 229.



Related Functions

`nwds_file_close` on page 43.

`nwds_file_copy` on page 46.

`nwds_file_create` on page 48.

`nwds_file_delete` on page 52.

`nwds_file_info` on page 55.

`nwds_file_open` on page 59.

`nwds_file_position` on page 63.

`nwds_file_read` on page 67.

`nwds_file_remove` on page 70.

`nwds_file_write` on page 76.







NWDS_FILE_WRITE

This function adds a new record to a file or a row to a table. The `nwds_file_write` function is provided as part of NetWeave's Data Server and/or Message Queue options.

```
NWDS_ERRNO nwds_file_write
    (NWDS_HANDLE      file_handle,
     NWDS_SIZE        buffer_size,
     void             *buffer,
     NWDS_ITEM_LIST   *control_items,
     NWDS_CALL_BACK   *call_back);
```

Parameter	Input	Output	Description
file_handle	✓		The handle returned from a call to <code>nwds_file_open</code> . NOTE: To use <code>nwds_file_open</code> , you must have the Data Server option.
buffer_size	✓		The number of bytes to be copied from the buffer (the next parameter, below). For a record-oriented file, if you try to exceed the maximum record size specified for the file, <code>NWDS_DATA_OVERFLOW</code> is returned. For a flat file, <code>buffer_size</code> bytes are appended to the file.
buffer	✓		The address of the array that contains the new data.
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to the callback structure. If <code>NULL</code> , the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

For a record-oriented file, the error `NWDS_DUPLICATE_KEY` is returned when you try to write a record that contains a key value that matches one for an index that does not permit duplicates. For more information about the return codes, see page 229.



Related Functions

`nwds_file_close` on page 43.

`nwds_file_copy` on page 46.

`nwds_file_create` on page 48.

`nwds_file_delete` on page 52.

`nwds_file_info` on page 55.

`nwds_file_open` on page 59.

`nwds_file_position` on page 63.

`nwds_file_read` on page 67.

`nwds_file_remove` on page 70.

`nwds_file_update` on page 73.





NWDS_INI_DELETE_NAME

This function, which is included in all NetWeave releases, deletes a token statement from an application's user group. The token `USER_NAME_GROUP` identifies the calling application's group in the INI file. You can also use `USER_NAME_GROUP = xxx` to supply a different INI file group for housing application-specific parameters. A token statement has the following syntax:

`token = value`

The `nwds_ini_delete_name` function does not modify the INI file, only the runtime contents of the user group.

```
NWDS_ERRNO nwds_ini_delete_name
(char          *token_name);
```

Parameter	Input	Output	Description
token_name	✓		The name of the token to remove from the user group. The name is a NULL-terminated string.

Return Code (output)

Return code	Description
NWDS_NAME_NOT_FOUND	An error occurred.
NWDS_SUCCESSFUL	The token was removed.

For more information about the return codes, see page 229.

Related Functions

`nwds_ini_get_name` on page 81.

`nwds_ini_put_name` on page 86.





NWDS_INI_GET_INT

This function, which is included in all NetWeave releases, returns the numeric value from the specified token statement from an application's user group. The token `USER_NAME_GROUP` identifies the user group in the application's root group. A token statement has the following syntax:

```
token = value
```

The `nwds_ini_get_int` function does not access the INI file, just the values from the runtime contents of the user group.

```
NWDS_ERRNO nwds_ini_get_int
(char      *lookup_name,
 int      *returned_value,
 int      default_value);
```

Parameter	Input	Output	Description
lookup_name	✓		The name of the token to be looked up. The name is a NULL-terminated string.
returned_value		✓	The token value.
default_value	✓		If the lookup_name is not found, the returned_value is set to default_value.

Return Code (output)

Return code	Description
NWDS_NAME_NOT_FOUND	An error occurred.
NWDS_BAD_PARAMETER	The <code>USER_NAME_GROUP</code> was not specified or was not found in the registered INI file
NWDS_DATA_OVERFLOW	Supplied buffer was not large enough to hold value.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_SUCCESSFUL	The token was found.

For more information about the return codes, see page 229.

Related Functions

`nwds_ini_delete_name` on page 79.



`nwds_ini_put_name` on page 86.





NWDS_INI_GET_NAME

This function, which is included in all NetWeave releases, returns the value from the specified token statement in an application's user group. The token `USER_NAME_GROUP` identifies the user group in the application's root group. A token statement has the following syntax:

```
token = value
```

The `nwds_ini_get_name` function does not access the INI file, just the values from the runtime contents of the user group.

```
NWDS_ERRNO nwds_ini_get_name
(char          *lookup_name,
NWDS_SIZE     buffer_size,
char          *returned_value);
```

Parameter	Input	Output	Description
lookup_name	✓		The name of the token to be looked up. The name is a NULL-terminated string.
buffer_size	✓		The maximum size allocated for the return string.
returned_value		✓	The returned value associated with the lookup_name.

Return Code (output)

Return code	Description
NWDS_NAME_NOT_FOUND	An error occurred. Desired name not found.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_BAD_PARAMETER	The <code>USER_NAME_GROUP</code> was not specified or was not found in the registered INI file
NWDS_DATA_OVERFLOW	Supplied buffer was not large enough to hold value.
NWDS_SUCCESSFUL	The token was found.

For more information about the return codes, see page 229.

Related Functions

`nwds_ini_delete_name` on page 79.

`nwds_ini_put_name` on page 86.







NWDS_INI_PUT_NAME

The `nwds_ini_put_name` function, which is included in all NetWeave releases, inserts a new token statement in the application's user group. The token `USER_NAME_GROUP` identifies the user group in the application's root group. A token statement has the following syntax:

```
token = value
```

The `nwds_ini_put_name` function does not access the INI file. Instead, it inserts new entries in the runtime contents of the user group.

```
NWDS_ERRNO nwds_ini_put_name
(char      *token,
char      *value);
```

Parameter	Input	Output	Description
token	✓		The token name is a NULL-terminated string.
value	✓		The value is a NULL-terminated string.

Return Code (output)

Return code	Description
NWDS_NO_MEMORY	The system is overloaded; process out of memory.
NWDS_SUCCESSFUL	The token statement was added.

For more information about the return codes, see page 229.

Related Functions

`nwds_ini_delete_name` on page 79.

`nwds_ini_get_name` on page 81.





NWDS_INIT

This must be the first NetWeave function call for any program that uses the NetWeave API. The default INI file name on platforms that support long file names is `netweave.ini`. The default for other file systems is `NWDSINI`. The default group (or section) name is `MAIN`. The `nwds_init` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_init
(char          *INI_file,
char          *INI_group);
```

Parameter	Input	Output	Description
INI_file	✓		The name (a NULL-terminated string) of the INI file for this program. If your application uses a sequence of INI files, this is the first one (called the root INI file) that NetWeave will use to resolve names and find communications parameters.
INI_group	✓		The name of the group in the root INI file where NetWeave begins its searches. The name is a NULL-terminated string.

Return Code (output)

Return code	Description
NWDS_FILE_NOT_FOUND	There is no INI file with this name.
NWDS_OPERATION_FAILED	The call did not complete successfully.
NWDS_SUCCESSFUL	The call completed successfully.

If `nwds_init` returns an error, it is important not to call `nwds_exit` (`nwds_init` cleans up after itself). For more information about the return codes, see page 229.

Related Functions

`nwds_item_load_char` on page 116.

`nwds_item_load_handle` on page 118.

`nwds_item_load_long` on page 120.

`nwds_item_load_short` on page 122.



NWDS_IPC_ACCEPT

To acknowledge that it has accepted a connection, the passive end of a connection (the server) calls `nwds_ipc_accept` to complete the setup of a connection with a client application. The `nwds_ipc_accept` function is included in all NetWeave releases.

If `nwds_ipc_accept` is called synchronously (i.e. `completion=NULL`), then the `client_handle` is valid when `NWDS_SUCCESSFUL` is returned. Otherwise the `client_handle` is undefined and invalid, but no further action is required to release the handle.

If `nwds_ipc_accept` is called asynchronously, the `client_handle` is not valid until the completion callback is returned with `NWDS_SUCCESSFUL`. If an error is returned from either `nwds_ipc_accept` or the callback function, then the handle is invalid, and it does not need to be cleaned up via `nwds_ipc_shutdown`.

Applications that communicate as peers may send unsolicited messages to each other at any time. To receive a message asynchronously whenever your partner sends one, you must specify the `data_received` callback.

```
NWDS_ERRNO nwds_ipc_accept
    (NWDS_HANDLE      server_handle,
     NWDS_HANDLE      *client_handle,
     NWDS_ITEM_LIST    *control_items,
     NWDS_CALL_BACK    *completion,
     NWDS_CALL_BACK    *data_received);
```

Parameter	Input	Output	Description
<code>server_handle</code>	✓		The handle that <code>nwds_ipc_publish</code> returns.
<code>client_handle</code>		✓	The identifier associated with the new connection. Use this identifier to write messages to the other application.
<code>control_items</code>	✓		A pointer to an array of system-specific parameters.
<code>completion</code>	✓		A pointer to a callback structure that contains the function that will be called when <code>nwds_ipc_accept</code> completes. If <code>NULL</code> , the call is synchronous.
<code>data_received</code>	✓		A pointer to the callback to be called each time the application receives a message from the other application. NOTE: If <code>data_received=NULL</code> , your application cannot receive unsolicited messages and must call <code>nwds_ipc_read</code> each time it is ready to process an incoming message. <code>nwds_ipc_read</code> will then suspend until data is available.



Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

For more information about the return codes, see page 229.

Related Functions

`nwds_session_close` on page 133.

`nwds_ipc_broadcast` on page 92.

`nwds_ipc_connect` on page 95.

`nwds_ipc_options` on page 98.

`nwds_ipc_publish` on page 101.

`nwds_ipc_read` on page 104.

`nwds_ipc_register` on page 107.

`nwds_ipc_shutdown` on page 110.

`nwds_ipc_write` on page 113.





NWDS_IPC_BROADCAST

This function broadcasts a message to applications that have called `nwds_ipc_register` to receive a message. The `nwds_ipc_broadcast` function completes when the message is sent, as opposed to `nwds_ipc_write`, which completes when the recipient receives the message. Because the sender does not know (or care) whether any receivers read the message, a broadcast does not require an acknowledgment. The `nwds_ipc_broadcast` function is provided as part of NetWeave's Broadcast option.

Applications that accept broadcast messages (receivers) register for broadcast messages by event type. When a sender broadcasts a message of a particular type, a receiver is interrupted only if it is a message of an event type for which the receiver has registered.

Broadcast messages are delivered to the network through a NetWeave Agent. For efficiency, certain resources associated with the session between the application and the agent are not released after each broadcast. These resources will be recovered when the application issues `nwds_session_close`.

NOTE: The `netweave.h` value `NWDS_MAX_USER_SIZE` determines the maximum length of a broadcast message.

```
NWDS_ERRNO nwds_ipc_broadcast
( char          *broadcast_port,
  NWDS_FILTER_CLASS  event_type,
  NWDS_SIZE         buffer_size,
  void            *buffer,
  NWDS_ITEM_LIST    *control_items,
  NWDS_CALL_BACK    *call_back );
```

Parameter	Input	Output	Description
broadcast_port	✓		The broadcast port is either a group in the INI file or a physical name whose first part is a NetWeave node name. If the port designates a group, the NAME statement in the group may identify a physical name that contains a node name. In either case, the node name designates a section name in the INI file that describes how to reach a NetWeave Agent that handles the broadcast.
event_type	✓		A parameter that allows receivers to identify messages as belonging to a class of messages known to the application.
buffer_size	✓		The length of the message in the buffer (next parameter, below). It may not exceed <code>NWDS_MAX_USER_SIZE</code> .
buffer	✓		The address of the array that contains the message.
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.



Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_session_close` on page 133.

`nwds_ipc_accept` on page 89.

`nwds_ipc_connect` on page 95.

`nwds_ipc_options` on page 98.

`nwds_ipc_publish` on page 101.

`nwds_ipc_read` on page 104.

`nwds_ipc_register` on page 107.

`nwds_ipc_shutdown` on page 110.

`nwds_ipc_write` on page 113.





NWDS_IPC_CONNECT

The initiating end of a connection (the client) calls `nwds_ipc_connect` to establish a connection with the server. This function is included in all NetWeave releases.

Applications that communicate as peers may send messages to each other at any time. To receive a message asynchronously whenever a partner sends one, you must specify the `data_received` callback. For more information, see page 89.

```
NWDS_ERRNO nwds_ipc_connect
(
    char          *server_name,
    NWDS_HANDLE   *server_id,
    NWDS_ITEM_LIST *control_items,
    NWDS_CALL_BACK *call_back,
    NWDS_CALL_BACK *data_received);
```

Parameter	Input	Output	Description
server_name	✓		The server name (a NULL-terminated string) is an INI file group that must contain one of the following: <ul style="list-style-type: none">• The token <code>PROTOCOL</code>, which contains information about the communications parameters required to form a connection with the server.• The token <code>NAME</code>, a physical name whose first part is a NetWeave node name. The node name indicates which section of the INI file describes how to reach a NetWeave Agent that mediates the connection with the application server.
server_id		✓	The identifier for the new session. Use this identifier to write messages to the other application.
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to a callback structure that contains the function that will be called when <code>nwds_ipc_connect</code> completes (when the <code>server_id</code> is returned). If <code>call_back=NULL</code> , the call is synchronous.
data_received	✓		A pointer to the callback to be called whenever a message is received from the other application. NOTE: If <code>data_received=NULL</code> , your application cannot receive unsolicited messages. It must call <code>nwds_ipc_read</code> each time it is ready to process an incoming message. <code>nwds_ipc_read</code> will then suspend until data is available.



Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_NO_MEMORY	The system is overloaded; process out of memory.
NWDS_LIBRARY_ERROR	Contact NetWeave support with the error traces and INI file for the program that received the error.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.

For more information about the return codes, see page 229.

Related Functions

`nwds_session_close` on page 133.

`nwds_ipc_accept` on page 89.

`nwds_ipc_broadcast` on page 92.

`nwds_ipc_options` on page 98.

`nwds_ipc_publish` on page 101.

`nwds_ipc_read` on page 104.

`nwds_ipc_register` on page 107.

`nwds_ipc_shutdown` on page 110.

`nwds_ipc_write` on page 113.





NWDS_IPC_OPTIONS

This function returns information (the number of queued messages, the length of each, a session's system type, etc.) from the communications layers. The `nwds_ipc_options` function is included in all NetWeave releases.

To specify what connection information you want to know, use the return item list. Before calling `nwds_ipc_options`, place the item types whose information you are requesting in the `returned_data` list structure. If the call is asynchronous, use persistent memory to receive the information that is returned. Do not declare return items on the local stack. Place the addresses of these locations in the `return_items` item list. When the function call completes, NetWeave copies the values of the requested items to the specified locations.

```
NWDS_ERRNO nwds_ipc_options
    (NWDS_HANDLE      ipc_handle,
     NWDS_ITEM_LIST    *returned_data);
```

Parameter	Input	Output	Description
<code>ipc_handle</code>	✓		The identifier associated with the session of interest.
<code>returned_data</code>		✓	The following parameters are supported on all platforms: <ul style="list-style-type: none">• <code>NWDS_IPC_ADDRESS</code>: protocol-dependent port address of the source of the first message in the queue (the message origin).• <code>NWDS_IPC_MAX_SEND_SIZE</code>: the maximum length of a message that can be sent on this virtual circuit.• <code>NWDS_IPC_MESSAGE_SIZE</code>: the length of the first message in the queue.• <code>NWDS_IPC_PROTOCOL</code>: the communications protocol used by the virtual circuit.• <code>NWDS_IPC_QUEUE_COUNT</code>: number of messages in the queue.• <code>NWDS_IPC_SYSTEM_TYPE</code>: the peer's system type.

Return Code (output)

Return code	Description
<code>NWDS_NOT_INITIALIZED</code>	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
<code>NWDS_BAD_HANDLE</code>	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

For more information about the return codes, see page 229.



Related Functions

`nwds_session_close` on page 133.

`nwds_ipc_accept` on page 89.

`nwds_ipc_broadcast` on page 92.

`nwds_ipc_connect` on page 95.

`nwds_ipc_publish` on page 101.

`nwds_ipc_read` on page 104.

`nwds_ipc_register` on page 107.

`nwds_ipc_shutdown` on page 110.

`nwds_ipc_write` on page 113.







NWDS_IPC_PUBLISH

The passive end of a connection (the server) calls `nwds_ipc_publish` to create the port to which a client will connect. The `nwds_ipc_publish` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_ipc_publish
(
    char          *public_name,
    NWDS_HANDLE   *server_handle,
    NWDS_ITEM_LIST *control_items,
    NWDS_CALL_BACK *call_back,
    NWDS_CALL_BACK *call_received);
```

Parameter	Input	Output	Description
public_name	✓		The name (a NULL-terminated string) in a process INI file group where communications layer information is stored. A public name never includes a NetWeave node name.
server_handle		✓	The handle associated with the public name. Use this handle to receive new connections from other applications.
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to a callback structure that contains the function that will be called when <code>nwds_ipc_publish</code> completes (when the <code>server_id</code> is returned). If NULL, the call is synchronous.
call_received	✓		A pointer to the callback that will be called each time the caller receives a new call from another application. If <code>call_received = NULL</code> , your application can handle requests from only one client at a time. To accept a connection from a remote client when <code>call_received = NULL</code> , you must do a synchronous <code>nwds_ipc_accept</code> followed by synchronous calls to <code>nwds_ipc_read</code> .

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.



Related Functions

`nwds_session_close` on page 133.

`nwds_ipc_accept` on page 89.

`nwds_ipc_broadcast` on page 92.

`nwds_ipc_connect` on page 95.

`nwds_ipc_options` on page 98.

`nwds_ipc_read` on page 104.

`nwds_ipc_register` on page 107.

`nwds_ipc_shutdown` on page 110.

`nwds_ipc_write` on page 113.







NWDS_IPC_READ

This function, which is included in all NetWeave releases, retrieves the first message from the queue or initiates a read request on the underlying communications channel. How this function works depends on whether your applications are peers or whether the client initiates all communication. In peer-to-peer communications, NetWeave queues any unsolicited messages until the application retrieves them. For more information, see the *NetWeave Programmer's Guide*.

Broadcasts are a special case of unsolicited messages. To retrieve a broadcast message, NetWeave uses the `ipc_handle` returned by `nwds_ipc_register`. The `nwds_ipc_read` function is always synchronous.

```
NWDS_ERRNO nwds_ipc_read
    (NWDS_HANDLE    ipc_handle,
     NWDS_SIZE      buffer_size,
     void           *buffer,
     NWDS_SIZE      *returned_size,
     NWDS_ITEM_LIST *control_items);
```

Parameter	Input	Output	Description
ipc_handle	✓		The session identifier that is returned from either <code>nwds_ipc_connect</code> (for the client), <code>nwds_ipc_accept</code> (for the server), or <code>nwds_ipc_register</code> (for broadcasts).
buffer_size	✓		The maximum number of bytes that may be copied to the buffer (next parameter). <code>NWDS_MAX_USER_SIZE</code> in <code>netweave.h</code> specifies the maximum length for a user's message. If <code>buffer_size</code> exceeds the <code>NWDS_MAX_USER_SIZE</code> value, NetWeave returns the error <code>NWDS_DATA_OVERFLOW</code> .
buffer	✓		The address of the array where the message is returned.
returned_size		✓	The actual number of bytes copied to the buffer.
control_items	✓		A pointer to an array of system-specific parameters. See Considerations on the next page.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

For more information about the return codes, see page 229.



Considerations

Client - Server Communication

If your application does not accept unsolicited messages, you must use `nwds_ipc_read` to initiate a read on the communications line. The call is blocked until a message is received. For more information about communication when one or the other of the applications does not accept unsolicited messages, please see the *NetWeave Programmer's Guide*.

control_items

Platform	Parameter	Description
All platforms	NWDS_IPC_CONVERT_NAME	A NULL-terminated string. If it is present, NetWeave will translate the message to the reader's format. The name is a group that contains information that describes each field in the structure of the message.
Tandem	NWDS_TAN_TAG	A long; Guardian I/O tag, output from read, input to write.

Related Functions

`nwds_session_close` on page 133.

`nwds_ipc_accept` on page 89.

`nwds_ipc_broadcast` on page 92.

`nwds_ipc_connect` on page 95.

`nwds_ipc_options` on page 98.

`nwds_ipc_publish` on page 101.

`nwds_ipc_register` on page 107.

`nwds_ipc_shutdown` on page 110.

`nwds_ipc_write` on page 113.





NWDS_IPC_REGISTER

This function registers an application to receive broadcasts of a specific event type. The `nwds_ipc_register` function is provided as part of NetWeave's Broadcast option.

Applications that accept broadcast messages (receivers) register for broadcast messages by event type. When a sender broadcasts a message, a receiver is interrupted only if it is a message of an event type for which the receiver has registered.

If the `data_received` callback is present, NetWeave collects and queues broadcast messages for delivery to the application. When another process broadcasts a message of the registered event type, NetWeave calls the `data_received` callback to notify the application that a new message has arrived. NetWeave queues the message until it is retrieved by a call to `nwds_ipc_read`. The same callback function may be associated with more than one event type. If there is no `data_received` callback, the application must call `nwds_ipc_read` to initiate the communications. In this case, `nwds_ipc_read` is a blocking call.

```
NWDS_ERRNO nwds_ipc_register
(
    char                *broadcast_port,
    NWDS_HANDLE         *ipc_handle,
    NWDS_FILTER_CLASS   event_type,
    NWDS_ITEM_LIST      *control_items,
    NWDS_CALL_BACK      *call_back,
    NWDS_CALL_BACK      *data_received);
```

Parameter	Input	Output	Description
broadcast_port	✓		The broadcast_port is either <ul style="list-style-type: none">• a physical name whose first part is a NetWeave node name• a group in the INI file (the group's NAME statement defines a physical name that contains a node name) The node name is the INI file section that describes how to reach the NetWeave Agent that receives the broadcast on behalf of the application.
ipc_handle		✓	When your application is ready to process a broadcast message, ipc_handle is passed to nwds_ipc_read to retrieve the message.
event_type	✓		A parameter that allows a receiver to identify messages as belonging to a class of messages known to the application.
control_items	✓		A pointer to an array of system-specific parameters. There are no control items in NetWeave version 2.0.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.
data_received	✓		A pointer to the callback structure that contains the function to call when the application receives a broadcast message.



Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_session_close` on page 133.

`nwds_ipc_accept` on page 89.

`nwds_ipc_broadcast` on page 92.

`nwds_ipc_connect` on page 95.

`nwds_ipc_options` on page 98.

`nwds_ipc_publish` on page 101.

`nwds_ipc_read` on page 104.

`nwds_ipc_shutdown` on page 110.

`nwds_ipc_write` on page 113.





NWDS_IPC_SHUTDOWN

This function terminates and cleans up a connection. The `nwds_ipc_shutdown` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_ipc_shutdown
    (NWDS_HANDLE      ipc_handle,
     NWDS_ITEM_LIST    *item_list,
     NWDS_CALL_BACK    *call_back);
```

Parameter	Input	Output	Description
ipc_handle	✓		The handle associated with a particular session that is returned from any of the following: <ul style="list-style-type: none">• <code>nwds_ipc_connect</code> (for the client)• <code>nwds_ipc_accept</code> (for the server)• <code>nwds_ipc_register</code> (for broadcasts)
item_list	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_SUCCESSFUL	The call completed successfully.

For more information about the return codes, see page 229.

Considerations

For `nwds_ipc_publish`, use `nwds_ipc_shutdown` to prevent any new calls from being accepted for a given public name.

For `nwds_ipc_register`, use `nwds_ipc_shutdown` to cancel a registration for broadcasts of a particular event type.



Related Functions

`nwds_session_close` on page 133.

`nwds_ipc_accept` on page 89.

`nwds_ipc_broadcast` on page 92.

`nwds_ipc_connect` on page 95.

`nwds_ipc_options` on page 98.

`nwds_ipc_publish` on page 101.

`nwds_ipc_read` on page 104.

`nwds_ipc_register` on page 107.

`nwds_ipc_write` on page 113.







NWDS_IPC_WRITE

This function, which is included in all NetWeave releases, sends a message through an established connection. To acknowledge that the message was received, the receiving application calls `nwds_ipc_read`. This is the only way to notify the sender that the message was delivered.

For peer-to-peer communication, either the application that initiates a connection (the client) or the application to which the connection is made (the server) may send one or more messages at any time to the partner application. For more information about peer-to-peer communication, see the *NetWeave Programmer's Guide*.

```
NWDS_ERRNO nwds_ipc_write
(
    NWDS_HANDLE    ipc_handle,
    NWDS_SIZE      buffer_size,
    void           *buffer,
    NWDS_ITEM_LIST *control_items,
    NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
ipc_handle	✓		The handle associated with a particular session that is returned by one of the following: <ul style="list-style-type: none">• <code>nwds_ipc_connect</code> (for the client)• <code>nwds_ipc_accept</code> (for the server)
buffer_size	✓		The number of bytes in the buffer (next parameter, below). In <code>netweave.h</code> , <code>NWDS_MAX_USER_SIZE</code> specifies the maximum length of a user's message. If this value exceeds <code>buffer_size</code> , NetWeave returns the error <code>NWDS_DATA_OVERFLOW</code> .
buffer	✓		The address of the message buffer.
control_items	✓		A pointer to an array of system-specific parameters. <code>NWDS_IPC_CONVERT_NAME</code> is the name (a NULL terminated string) of a group that contains information about each field in the structure of the message. If this named group is present, NetWeave will translate the message to a common network format.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.



Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

For more information about the return codes, see page 229.

Related Functions

`nwds_session_close` on page 133.

`nwds_ipc_accept` on page 89.

`nwds_ipc_broadcast` on page 92.

`nwds_ipc_connect` on page 95.

`nwds_ipc_options` on page 98.

`nwds_ipc_publish` on page 101.

`nwds_ipc_read` on page 104.

`nwds_ipc_register` on page 107.

`nwds_ipc_shutdown` on page 110.





NWDS_ITEM_LOAD_CHAR

This function, which is included in all NetWeave releases, assigns a char array to an item list element. Programming languages such as COBOL that do not support pointer data types can use the `nwds_item_load_char` function.

```
NWDS_ERRNO nwds_item_load_char (  
    NWDS_ITEM_LIST    *control_list,  
    NWDS_SIZE          index,  
    NWDS_ITEM_TYPE     item_type,  
    NWDS_SIZE          item_size,  
    void               *item_value);
```

Parameter	Input	Output	Description
control_list	✓		An item list array you want to construct.
index	✓		The index of the array element you want to load.
item_type	✓		The type of the item. Select from the list in <code>netweave.h</code> .
item_size	✓		The size of the char input array, which is added to the control list.
item_value	✓		The address of the first element of the char array.

Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	The call completed successfully.
NWDS_INVALID_ITEM	An item in an item list is not of the proper data type or the value is out of range.

For more information about the return codes, see page 229.

Related Functions

`nwds_init` on page 88.

`nwds_item_load_handle` on page 118.

`nwds_item_load_long` on page 120.

`nwds_item_load_short` on page 122.





NWDS_ITEM_LOAD_HANDLE

This function, which is included in all NetWeave releases, assigns a handle value (a transaction identifier) to an item list element. Programming languages such as COBOL that do not support pointer data types can use the `nwds_item_load_handle` function.

```
NWDS_ERRNO nwds_item_load_handle (  
    NWDS_ITEM_LIST    *control_list,  
    NWDS_SIZE          index,  
    NWDS_ITEM_TYPE     item_type,  
    NWDS_HANDLE        *item_value);
```

Parameter	Input	Output	Description
control_list	✓		An item list array you want to construct.
index	✓		The index of the array element you want to load.
item_type	✓		The type of the item. (Select from the list in <code>netweave.h</code>).
item_value	✓		The name of the variable in the program's data space whose value is added to the list. NOTE: This name must be a NetWeave handle.

Return Code (output)

Return code	Description
NWDS_INVALID_ITEM	An item in an item list is not of the proper data type or the value is out of range.
NWDS_SUCCESSFUL	The call completed successfully.

For more information about the return codes, see page 229.

Related Functions

`nwds_init` on page 88.

`nwds_item_load_char` on page 116.

`nwds_item_load_long` on page 120.

`nwds_item_load_short` on page 122.





NWDS_ITEM_LOAD_LONG

This function, which is included in all NetWeave releases, assigns a long integer value to an item list element. Programming languages such as COBOL that do not support pointer data types can use the `nwds_item_load_long` function.

```
NWDS_ERRNO nwds_item_load_long (  
    NWDS_ITEM_LIST    *control_list,  
    NWDS_SIZE          index,  
    NWDS_ITEM_TYPE     item_type,  
    long               *item_value);
```

Parameter	Input	Output	Description
control_list	✓		An item list array you want to create.
index	✓		The index of the array element you want to load.
item_type	✓		The type of the item. (Select from the list in <code>netweave.h</code> .)
item_value	✓		The name of the variable in the program's data space whose value is added to the list.

Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	The call completed successfully.
NWDS_INVALID_ITEM	An item in an item list is not of the proper data type or the value is out of range.

For more information about the return codes, see page 229.

Related Functions

`nwds_init` on page 88.

`nwds_item_load_char` on page 116.

`nwds_item_load_handle` on page 118.

`nwds_item_load_short` on page 122.





NWDS_ITEM_LOAD_SHORT

This function, which is included in all NetWeave releases, assigns a short integer value to an item list element. Programming languages such as COBOL that do not support pointer data types can use the `nwds_item_load_short` function.

```
NWDS_ERRNO nwds_item_load_short (  
    NWDS_ITEM_LIST    *control_list,  
    NWDS_SIZE          index,  
    NWDS_ITEM_TYPE     item_type,  
    short              *item_value);
```

Parameter	Input	Output	Description
control_list	✓		An item list array you want to construct.
index	✓		The index of the array element you want to load.
item_type	✓		The type of the item. (Select from the list in <code>netweave.h</code>).
item_value	✓		The name of the short variable in the program's data space whose value is added to the list.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_SUCCESSFUL	The call completed successfully.
NWDS_INVALID_ITEM	An item in an item list is not of the proper data type, or the value is out of range.

For more information about the return codes, see page 229.

Related Functions

`nwds_init` on page 88.

`nwds_item_load_char` on page 116.

`nwds_item_load_handle` on page 118.

`nwds_item_load_long` on page 120.





NWDS_LOGOFF

Use this function to exit from a remote system. The `nwds_logoff` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_logoff
( char          *system_name,
  NWDS_ITEM_LIST *control_items,
  NWDS_CALL_BACK *call_back );
```

Parameter	Input	Output	Description
system_name	✓		The name of the system from which you want to disconnect.
control_items	✓		A pointer to an array of system-specific parameters. There are no control_items currently defined for nwds_logoff.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_logon` on page 126.

`nwds_msglog` on page 126.

`nwds_password` on page 130.





NWDS_LOGON

This function, which is included in all NetWeave releases, connects the calling application to a remote system with validated access. To use the `nwds_logon` function, you must have a NetWeave Agent running on the remote platform. The Agent validates the supplied user name/password against the remote platform's authentication facility. For additional security, the calling application may also request that the Agent on the remote system periodically send a challenge string.

```
NWDS_ERRNO nwds_logon
( char          *system_name,
  NWDS_SIZE     name_size,
  void          *user_name,
  NWDS_ITEM_LIST *control_items,
  NWDS_ITEM_LIST *return_items,
  NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
system_name	✓		The name of the system to which you want to connect.
name_size	✓		The size of the name.
user_name	✓		The name of the user who is logging on.
control_items	✓		A pointer to an array of system-specific parameters. There are no control_items currently defined for <code>nwds_logon</code> .
return_items		✓	If you implement challenge-response authentication, you may request the return item <code>nwds_acl_challenge</code> .
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_logoff` on page 124.

`nwds_msglog` on page 126.

`nwds_password` on page 130.



NWDS_MSGLOG

The `nwds_msglog` function allows applications to add application-level logging to the NetWeave logging facility, and to control the logging level (DEBUG, INFO, ERROR) external to the operating program. The application must set the severity level of a message to a value specified by `nwds_msglog_severity`. The messages in the log will be formatted according to the conventions of the `printf` function in C.

The `nwds_msglog` function is always synchronous. For more information, see “Message and Error Logging Considerations” in the *NetWeave Configuration Guide*.

```
typedef enum
{
    NWDS_MLSTRACE = 1,    /*"TRACE" msglog_level*/
    NWDS_MLSINFO,        /*"INFO" msglog_level*/
    NWDS_MLSWARNING,     /*"WARNING" msglog_level*/
    NWDS_MLSERROR,       /*"ERROR" msglog_level*/
    NWDS_MLSFATAL        /*"FATAL" msglog_level*/
} NWDS_MSGLOG_SEVERITY;

NWDS_ERRNO nwds_msglog(
    NWDS_MSGLOG_SEVERITY severity,
    const char          *format,
    ...                  parameters...);
```

Parameter	Input	Output	Description
severity	✓		One of the values specified by the enumerated type <code>nwds_msglog_severity</code> , cf. <code>netweave.h</code> . Indicates the level of error associated with the generated message.
format	✓		The format string follows the conventions of the <code>printf</code> function in C.
parameters	✓		The data types and order of parameters must match the data type indicators in the format string.

Return Code

Return code	Description
NWDS_BAD_PARAMETER	You are trying to call a function but one of your parameters is out of range.

For more information about the return codes, see page 229.



Related Functions

`nwds_logoff` on page 124.

`nwds_logon` on page 126.

`nwds_password` on page 130.







NWDS_PASSWORD

This function, which is included in all NetWeave releases, sends a password to a remote system. (As an option, you may return a response to a challenge to the remote server.)

```
NWDS_ERRNO nwds_password
    (char          *system_name,
     NWDS_SIZE     buffer_size,
     void          *password,
     NWDS_ITEM_LIST *control_item,
     NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
system_name	✓		The name of the system to which you want to connect.
buffer_size	✓		The length of the password string.
password	✓		A pointer to an array that contains the password.
control_items	✓		A pointer to an array of system-specific parameters. Currently there are no control items specific to nwds_password.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling nwds_init() and before calling nwds_exit(). Call nwds_init() and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

nwds_logoff on page 124.

nwds_logon on page 126.

nwds_msglog on page 126.



NWDS_PING

The `nwds_ping` function determines the status of a connection to a NetWeave Agent by sending a (very short) message to the NetWeave Agent, which echoes a return message. If there is no connection, one is created if it's possible. If `nwds_system_type` has already established a connection, it does not send a message to the NetWeave Agent. The `nwds_ping` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_ping
( char          *netweave_node,
  NWDS_ITEM_LIST *item_list,
  NWDS_CALL_BACK *call_back );
```

Parameter	Input	Output	Description
<code>netweave_node</code>	✓		A section name in the INI file that specifies the parameters for the communications layer between your application and the application or NetWeave Server that corresponds to the node. The name is a sequence of letters followed by two colons and terminated with a NULL byte. It may be either a logical name or a node name.
<code>item_list</code>	✓		A pointer to an array of system-specific parameters. Currently there are no item types specific for <code>nwds_ping</code> .
<code>call_back</code>	✓		A pointer to a callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
<code>NWDS_NOT_INITIALIZED</code>	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_system_type` on page 166.





NWDS_SESSION_CLOSE

The `nwds_session_close` function releases all resources associated with a NetWeave node, and is used mainly to clean up resources associated with a broadcast session. This function is included in all NetWeave releases. For more information, see `nwds_ipc_broadcast` on page 92.

```
NWDS_ERRNO nwds_session_close
    (char          *netweave_node,
     NWDS_ITEM_LIST *item_list,
     NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
<code>netweave_node</code>	✓		A section name in the INI file that specifies the parameters for the communications layer between your application and the application or NetWeave Agent corresponding to the node. The name is a sequence of letters followed by two colons and terminated with a NULL byte. The name may be either a logical name or a node name.
<code>item_list</code>	✓		A pointer to an array of system-specific parameters. Currently no item types are specific to <code>nwds_session_close</code> .
<code>call_back</code>	✓		A pointer to a callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
<code>NWDS_NOT_INITIALIZED</code>	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
<code>NWDS_PROCESS_NOT_CONNECTED</code>	The session is already closed.

For more information about the return codes, see page 229.

Related Functions

`nwds_ipc_accept` on page 89.

`nwds_ipc_broadcast` on page 92.

`nwds_ipc_connect` on page 95.

`nwds_ipc_options` on page 98.



`nwds_ipc_publish` on page 101.

`nwds_ipc_read` on page 104.

`nwds_ipc_register` on page 107.

`nwds_ipc_shutdown` on page 110.

`nwds_ipc_write` on page 113.







NWDS_SLEEP

A call to `nwds_sleep` suspends your application, and the item types you specify further define if and when your process resumes execution. The `nwds_sleep` function is included in all NetWeave releases.

Programs containing asynchronous operations call `nwds_sleep` to resynchronize. Either your program waits indefinitely while callbacks continue processing, or your program waits for an event to occur or for a timeout period to expire.

```
NWDS_ERRNO nwds_sleep
    (NWDS_MILLISECONDS timeout,
     NWDS_ITEM_LIST      *item_list,
     NWDS_CALL_BACK      *call_back);
```

Parameter	Input	Output	Description
timeout	✓		If positive, the timeout duration is specified in milliseconds. If timeout=-1, timeout is forever.
item_list	✓		A pointer to an array of system-specific parameters. The following item types apply to <code>nwds_sleep</code> : <ul style="list-style-type: none">• NWDS_KERNEL_ONCE• NWDS_KERNEL_SUSPEND For more information, see Considerations below.
call_back	✓		<code>nwds_sleep</code> ignores the <code>call_back</code> parameter. Set it to NULL.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_PENDING	The operation was initiated successfully. Final status and data will be delivered to the specified callback function.
NWDS_SUCCESSFUL	The call completed successfully.

For more information about the return codes, see page 229.



Considerations

When using `nwds_sleep`, the term *next event* refers to any activity that causes a user's callback to be called - typically activity in a communications layer controlled by the NetWeave middleware. Your program may suspend in any of the following ways:

Timeout value	Item type	Result
Negative (-1)	NWDS_KERNEL_ONCE	Your program will wait forever for the next event. Applications may use this form of <code>nwds_sleep</code> to construct a poll loop that checks whether some application activity has occurred, and if not, suspends again.
	NWDS_KERNEL_ONCE is not present	Your program will suspend indefinitely. Use this form of <code>nwds_sleep</code> for normal asynchronous operations where all application work is performed in the callbacks in reaction to messages, broadcasts, or triggers that the program receives. This is referred to as the <i>event-driven</i> model of asynchronous programming.
Positive	NWDS_KERNEL_SUSPEND	The underlying software will continue to call callback functions until the timer expires. Use this form of <code>nwds_sleep</code> to introduce a delay in your main line of execution, while continuing to allow callback functions to be invoked by the NetWeave system. For example, use this form of <code>nwds_sleep</code> for application level poll loops where multiple events must be resynchronized.
	NWDS_KERNEL_SUSPEND is not present	The underlying software will suspend until the next event, such as the expiration of the timeout period. If the next event is not expiration of the timer, but instead (for example) some communications activity, the timer is canceled and control is returned to your program. Use this form of <code>nwds_sleep</code> to implement application-level timeouts for asynchronous function calls.

Related Functions

`nwds_exit` on page 41.

`nwds_sleep_callback` on page 138.

`nwds_sleep_clear_callback` on page 141.

`nwds_stop` on page 164.

`nwds_timer_start` on page 169.

`nwds_timer_stop` on page 172.



NWDS_SLEEP_CALLBACK

The `nwds_sleep_callback` function lets you enqueue a callback from a notification routine to permit serialization of user callbacks. NetWeave then calls the enqueued callback during the next processing cycle. The `nwds_sleep_callback` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_sleep_callback
    (NWDS_CALL_BACK    *call_back,
     NWDS_ERRNO        errno);
```

Parameter	Input	Output	Description
call_back	✓		The callback structure identifies the application function that NetWeave will call at the beginning of the next processing cycle. This memory must persist after the current function exits.
errno	✓		The condition code that the enqueueing function wants to pass to the subsequent callback procedure.

Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	The call completed successfully.
NWDS_NO_MEMORY	The application exceeds the available heap space.

For more information about the return codes, see page 229.

Considerations


To serialize callback processing, use `nwds_sleep_callback` to enqueue a callback from a notification routine. You must enter the sleep forever loop with a special item list type called `NWDS_KERNEL_LOOP`. The code fragment below shows how to do this.

```
NWDS_ITEM_LIST  items[2];

    items[0].type = NWDS_KERNEL_LOOP;
    items[1].type = NWDS_END_OF_LIST;

    status = nwds_sleep( -1L, items, NULL);
```

During one processing cycle, additional callbacks may be enqueued. Any callback that is enqueued during a cycle is invoked at the beginning of the next cycle. If several callbacks are queued at the same



time, they are invoked in the chronological order in which they were enqueued. If an application is waiting under the kernel suspend option, `nwds_sleep` will return at the end of the cycle.

Related Functions

`nwds_exit` on page 41.

`nwds_sleep` on page 136.

`nwds_sleep_clear_callback` on page 141.

`nwds_stop` on page 164.

`nwds_timer_start` on page 169.

`nwds_timer_stop` on page 172.





NWDS_SLEEP_CLEAR_CALLBACK

This function clears or cancels a previously enqueued callback, and should be used in conjunction with the `nwds_sleep_callback` function. The `nwds_sleep_clear_callback` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_sleep_clear_callback  
    (NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
call_back	✓		The callback structure identifies the callback that NetWeave is to cancel.

Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	The call completed successfully.
NWDS_PROCESS_NOT_FOUND	The callback is not currently registered with NetWeave.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.

For more information about the return codes, see page 229.

Related Functions

`nwds_exit` on page 41.

`nwds_sleep` on page 136.

`nwds_sleep_callback` on page 138.

`nwds_stop` on page 164.

`nwds_timer_start` on page 169.

`nwds_timer_stop` on page 172.





NWDS_SQL_COLUMN_BIND

This function relates a variable in the user's data space to a specific column of a SQL table. The `nwds_sql_column_bind` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_sql_column_bind
    (NWDS_HANDLE      handle,
     NWDS_SIZE        column_number,
     void              *value_address);
```

Parameter	Input	Output	Description
handle	✓		The handle to use for operations on this table.
column_number	✓		The index of the column.
value_address	✓		The name of a variable in the program's data space to which data from this column will be returned.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_SQL_INVALID_COLUMN	Invalid column index.
NWDS_SUCCESSFUL	The bind is successful.

For more information about the return codes, see page 229.

Related Functions

`nwds_sql_column_count` on page 143.

`nwds_sql_column_get` on page 146.

`nwds_sql_column_info` on page 149.

`nwds_sql_connect` on page 152.

`nwds_sql_disconnect` on page 166.

`nwds_sql_execute` on page 154.

`nwds_sql_fetch` on page 158.

`nwds_sql_select` on page 161.



NWDS_SQL_COLUMN_COUNT

After successful execution of a `SELECT` command, use `nwds_sql_column_count` to obtain the number of columns described by the `SELECT`. This function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_sql_column_count
    (NWDS_HANDLE      handle,
     NWDS_SIZE        *column_number);
```

Parameter	Input	Output	Description
handle	✓		The handle to use for operations on this file.
column_number		✓	The number of columns described.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_SUCCESSFUL	The bind is successful.

For more information about the return codes, see page 229.

Related Functions

`nwds_sql_column_bind` on page 143.

`nwds_sql_column_get` on page 146.

`nwds_sql_column_info` on page 149.

`nwds_sql_connect` on page 152.

`nwds_sql_disconnect` on page 166.

`nwds_sql_execute` on page 154.

`nwds_sql_fetch` on page 158.

`nwds_sql_select` on page 161.





NWDS_SQL_COLUMN_GET

This function returns the name of a column as a string. The `nwds_sql_column_get` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_sql_column_get
    (NWDS_HANDLE      handle,
     NWDS_SIZE        column_number,
     NWDS_SIZE        maximum_name_size,
     void             *value_address,
     NWDS_SIZE        *actual_name_size);
```

Parameter	Input	Output	Description
handle	✓		The handle to use for operations on this file.
column_number	✓		The index of the column.
maximum_name_size	✓		The maximum size of the column name.
value_address	✓		The name of a variable in the program's data space to which data from this column will be returned.
actual_name_size		✓	The actual size of the column name.

Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	The bind is successful.
NWDS_BAD_HANDLE	The first parameter is invalid. You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_SQL_INVALID_COLUMN	The second parameter is invalid.
NWDS_DATA_OVERFLOW	The third parameter is too small.

For more information about the return codes, see page 229.

Related Functions

`nwds_sql_column_bind` on page 143.

`nwds_sql_column_count` on page 143.

`nwds_sql_column_info` on page 149.



`nwds_sql_connect` on page 152.

`nwds_sql_disconnect` on page 166.

`nwds_sql_execute` on page 154.

`nwds_sql_fetch` on page 158.

`nwds_sql_select` on page 161.







NWDS_SQL_COLUMN_INFO

After a successful execution of a `SELECT` command, use `nwds_sql_column_info` to obtain the name and data type of each of the columns described by `SELECT`. This function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_sql_column_info
    (NWDS_HANDLE      handle,
     NWDS_SIZE        column_number,
     NWDS_SIZE        name_buffer,
     char             *name,
     NWDS_SIZE        *return_size,
     NWDS_DATA_CLASS  *data_type);
```

Parameter	Input	Output	Description
handle	✓		The handle to use for operations on this file.
column_number	✓		The number of the column for which you wish to obtain information.
name_buffer	✓		The size of the buffer for name.
name		✓	The name of the data type.
return_size		✓	The size of the column name.
data_type		✓	The data type of the column.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_DATA_OVERFLOW	The third parameter is too small.
NWDS_SQL_INVALID_COLUMN	The second parameter is invalid.
NWDS_SUCCESSFUL	The bind is successful.

For more information about the return codes, see page 229.



Related Functions

`nwds_sql_column_bind` on page 143.

`nwds_sql_column_count` on page 143.

`nwds_sql_column_get` on page 146.

`nwds_sql_connect` on page 152.

`nwds_sql_disconnect` on page 166.

`nwds_sql_execute` on page 154.

`nwds_sql_fetch` on page 158.

`nwds_sql_select` on page 161.







NWDS_SQL_CONNECT

Use this function to attach to the SQL database at the system specified by `system_name`. The `nwds_sql_connect` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_sql_connect
    (char          *system_name,
     NWDS_HANDLE   *handle,
     NWDS_ITEM_LIST *control_items,
     NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
system_name	✓		The name of the system to which you want to connect.
handle		✓	The handle to use for subsequent operations on this file.
control_items	✓		A pointer to an array of system-specific parameters: <ul style="list-style-type: none">• NWDS_SQL_USERNAME• NWDS_SQL_PASSWORD
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_sql_column_bind` on page 143.

`nwds_sql_column_count` on page 143.

`nwds_sql_column_get` on page 146.

`nwds_sql_column_info` on page 149.

`nwds_sql_disconnect` on page 166.

`nwds_sql_execute` on page 154.

`nwds_sql_fetch` on page 158.

`nwds_sql_select` on page 161.





NWDS_SQL_DISCONNECT

Use this function to disconnect from a SQL database. The `nwds_sql_disconnect` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_sql_disconnect
    (NWDS_HANDLE      handle,
     NWDS_ITEM_LIST    *control_items,
     NWDS_CALL_BACK    *call_back);
```

Parameter	Input	Output	Description
handle	✓		The handle to use for operations on this file.
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_sql_column_bind` on page 143.

`nwds_sql_column_count` on page 143.

`nwds_sql_column_get` on page 146.

`nwds_sql_column_info` on page 149.

`nwds_sql_connect` on page 152.

`nwds_sql_execute` on page 154.

`nwds_sql_fetch` on page 158.

`nwds_sql_select` on page 161.



NWDS_SQL_EXECUTE

Use this function to execute a command on a remote SQL database. The `nwds_sql_execute` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_sql_execute
(
    NWDS_HANDLE    handle,
    char            *command,
    char            **variable_values,
    NWDS_ITEM_LIST  *control_items,
    NWDS_ITEM_LIST  *return_items,
    NWDS_CALL_BACK  *call_back);
```

Parameter	Input	Output	Description
handle	✓		The handle to use for operations on this file.
command	✓		The SQL command to execute during the call. Not all SQL statements will be processed by <code>nwds_sql_execute</code> . For example, SELECT and FETCH statements cannot be executed; but they have their own API. For Tandem's NonStopSQL, the BEGIN WORK, COMMIT WORK and ROLLBACK WORK commands are rejected. Instead, use <code>nwds_tp_start</code> , <code>nwds_tp_commit</code> , and <code>nwds_tp_abort</code> .
variable_values	✓		An array of strings that specify input values for parameters in the command.
control_items	✓		A pointer to an array of system-specific parameters.
return_items		✓	A pointer to an array of system-specific parameters: <ul style="list-style-type: none">• NWDS_SQL_ERROR_CODE: the host SQL process returns these codes.• NWDS_SQL_ERROR_TEXT: The programmer must interpret the codes for the platform and determine the nature of the problem.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.



Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

If `nwds_sql_execute` returns `nwds_sql_invalid_verb`, use another NetWeave function for this task. For more information about the return codes, see page 229.

Related Functions

`nwds_sql_column_bind` on page 143.

`nwds_sql_column_count` on page 143.

`nwds_sql_column_get` on page 146.

`nwds_sql_column_info` on page 149.

`nwds_sql_connect` on page 152.

`nwds_sql_disconnect` on page 166.

`nwds_sql_fetch` on page 158.

`nwds_sql_select` on page 161.





NWDS_SQL_FETCH

Use this function to return one row and store the values in the user's data space. The `nwds_sql_fetch` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_sql_fetch
(
    NWDS_HANDLE      handle,
    NWDS_ITEM_LIST   *control_items,
    NWDS_ITEM_LIST   *return_items,
    NWDS_CALL_BACK   *call_back);
```

Parameter	Input	Output	Description
handle	✓		The handle to use for operations on this file.
control_items	✓		A pointer to an array of system-specific parameters.
return_items		✓	A pointer to an array of system-specific parameters: <ul style="list-style-type: none">NWDS_SQL_ERROR_CODE: the host SQL process returns these codes.NWDS_SQL_ERROR_TEXT: the programmer must interpret the codes for the platform and determine the nature of the problem.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

For more information about the return codes, see page 229.

Related Functions

`nwds_sql_column_bind` on page 143.

`nwds_sql_column_count` on page 143.

`nwds_sql_column_get` on page 146.

`nwds_sql_column_info` on page 149.

`nwds_sql_connect` on page 152.



`nwds_sql_disconnect` on page 166.

`nwds_sql_execute` on page 154.

`nwds_sql_select` on page 161.







NWDS_SQL_SELECT

This function executes `SELECT` on a remote SQL database. The `nwds_sql_select` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_sql_select
(
    NWDS_HANDLE      handle,
    char             *command,
    char             **variable_values,
    NWDS_ITEM_LIST   *control_items,
    NWDS_ITEM_LIST   *return_items,
    NWDS_CALL_BACK   *call_back);
```

Parameter	Input	Output	Description
handle	✓		The handle to use for subsequent operations on this file.
command	✓		The SQL select command.
variable_values	✓		An array of strings that specify input values for parameters in the command.
control_items	✓		A pointer to an array of system-specific parameters.
return_items		✓	A pointer to an array of system-specific parameters.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

For more information about the return codes, see page 229.

Related Functions

`nwds_sql_column_bind` on page 143.

`nwds_sql_column_count` on page 143.

`nwds_sql_column_get` on page 146.

`nwds_sql_column_info` on page 149.

`nwds_sql_connect` on page 152.



`nwds_sql_disconnect` on page 166.

`nwds_sql_execute` on page 154.

`nwds_sql_fetch` on page 158.







NWDS_STOP

This function, which is included in all NetWeave releases, stops the execution of a remote process that you started.

```
NWDS_ERRNO nwds_stop
    (char          *process_name,
     NWDS_ITEM_LIST *item_list,
     NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
process_name	✓		The name of the process (system-specific). To retrieve the name of a process that you started, see <code>nwds_execute</code> on page 36.
item_list	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to the callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_exit` on page 41.

`nwds_sleep` on page 136.

`nwds_sleep_callback` on page 138.

`nwds_sleep_clear_callback` on page 141.

`nwds_timer_start` on page 169.

`nwds_timer_stop` on page 172.





NWDS_SYSTEM_TYPE

This function determines the system type on which the designated NetWeave Agent or remote application is running. If the system type has already been determined, `nwds_system_type` returns the previously obtained system type information without sending a message to the remote system. The `nwds_system_type` function is included in all NetWeave releases.

NOTE: Unlike `nwds_system_type`, the `nwds_ping` function always sends a message to the NetWeave Agent.

```
NWDS_ERRNO nwds_system_type
(char          *netweave_node,
NWDS_SYSTEM_CLASS *system_type,
NWDS_ITEM_LIST  *control_items,
NWDS_CALL_BACK  *call_back);
```

Parameter	Input	Output	Description
netweave_node	✓		A section name in the INI file that specifies the parameters for the communications layer between your application and the application or NetWeave Agent that corresponds to the node. The name is a sequence of letters followed by two colons and terminated with a NULL byte. The name may be either a logical name or a node name.
system_type		✓	For the complete list of supported system types, see <code>netweave.h</code> .
control_items	✓		A pointer to an array of system-specific parameters. Currently there are no items types specific to <code>nwds_system_type</code> .
call_back	✓		A pointer to a callback structure. If NULL, the call is synchronous

Return Code (output)

Return code	Description
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_SUCCESSFUL	The call completed successfully.

For more information about the return codes, see page 229.



Related Functions

`nwds_ping` on page 131.







NWDS_TIMER_START

The `nwds_timer_start` function associates a callback with a timer event (identified by a `nwds_handle`) that is returned to the user. When the timer expires, NetWeave calls the user's callback function. The `nwds_timer_start` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_timer_start
    (NWDS_MILLISECONDS timer_value,
     NWDS_HANDLE        *handle,
     NWDS_ITEM_LIST     *control_items,
     NWDS_CALL_BACK     *call_back);
```

Parameter	Input	Output	Description
timer_value	✓		Timeout value, in milliseconds.
handle		✓	The identifier to use for subsequent timer operations.
control_items	✓		A pointer to an array of system-specific parameters: <ul style="list-style-type: none">• <code>NWDS_TIMER_TYPE</code>: a logical value that defines the timer type.• <code>NWDS_PERSISTENT</code>: automatically resets the timer after each expiration.• <code>NWDS_SINGLE</code> (default): a “one-shot” timer that is set once and either expires or is stopped. This form of timer is intended for applications that require a regular signal.
call_back	✓		A pointer to a callback structure.

Return Code (output)

Return code	Description
<code>NWDS_BAD_PARAMETER</code>	You are trying to call a function but one of your parameters is out of range.
<code>NWDS_NOT_INITIALIZED</code>	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.



Related Functions

`nwds_exit` on page 41.

`nwds_sleep` on page 136.

`nwds_sleep_callback` on page 138.

`nwds_sleep_clear_callback` on page 141.

`nwds_stop` on page 164.

`nwds_timer_stop` on page 172.







NWDS_TIMER_STOP

This function halts the timer associated with a handle. It is included in all NetWeave releases.

```
NWDS_ERRNO nwds_timer_stop (NWDS_HANDLE);
```

Parameter	Input	Output	Description
handle	✓		The identifier returned from a call to <code>nwds_timer_start</code> .

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	There is no timer associated with the handle.
NWDS_BAD_PARAMETER	You are trying to call a function but one of your parameters is out of range.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_SUCCESSFUL	The timer was stopped successfully.

For more information about the return codes, see page 229.

Related Functions

`nwds_exit` on page 41.

`nwds_sleep` on page 136.

`nwds_sleep_callback` on page 138.

`nwds_sleep_clear_callback` on page 141.

`nwds_stop` on page 164.

`nwds_timer_start` on page 169.





NWDS_TP_ABORT

This function aborts a transaction. It is included in all NetWeave releases.

```
NWDS_ERRNO nwds_tp_abort
    (NWDS_HANDLE      tp_handle,
     NWDS_ITEM_LIST    *control_items,
     NWDS_CALL_BACK    *call_back);
```

Parameter	Input	Output	Description
tp_handle	✓		The handle returned from nwds_tp_start.
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to a callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_NO_MEMORY	The system is overloaded; process out of memory.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling nwds_init() and before calling nwds_exit(). Call nwds_init() and re-issue the NetWeave function call.
NWDS_PENDING	The operation has been initiated successfully. Final status and data will be delivered to the specified callback function.
NWDS_SUCCESSFUL	The call completed successfully.

For more information about the return codes, see page 229.

Related Functions

nwds_tp_commit on page 175.

nwds_tp_resume on page 176.

nwds_tp_start on page 177.

nwds_tp_status on page 180.



NWDS_TP_COMMIT

This function commits a transaction. Any file changes associated with the transaction identifier are made at this point and become visible to other users. The `nwds_tp_commit` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_tp_commit
    (NWDS_HANDLE      tp_handle,
     NWDS_ITEM_LIST    *control_items,
     NWDS_CALL_BACK    *call_back);
```

Parameter	Input	Output	Description
tp_handle	✓		The handle returned from <code>nwds_tp_start</code> .
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to a callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_NO_MEMORY	The system is overloaded; process out of memory.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_PENDING	The operation has been initiated successfully. Final status and data will be delivered to the specified callback function.

For more information about the return codes, see page 229.

Related Functions

`nwds_tp_abort` on page 174.

`nwds_tp_resume` on page 176.

`nwds_tp_start` on page 177.

`nwds_tp_status` on page 180.



NWDS_TP_RESUME

Use this function, which is included in all NetWeave releases, to resume a transaction. The `nwds_tp_resume` function applies only to Tandem.

```
NWDS_ERRNO nwds_tp_resume
    (NWDS_HANDLE      tp_handle,
     NWDS_ITEM_LIST    *control_items,
     NWDS_CALL_BACK    *call_back);
```

Parameter	Input	Output	Description
tp_handle	✓		The handle returned from <code>nwds_tp_start</code> .
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to a callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_tp_abort` on page 174.

`nwds_tp_commit` on page 175.

`nwds_tp_start` on page 177.

`nwds_tp_status` on page 180.





NWDS_TP_START

This function starts a transaction at a remote host that supports a transaction protection (TP) monitor. On Tandem, if NetWeave handles any of the I/O, an application must call `nwds_tp_start` instead of the Guardian `begintransaction`. The `nwds_tp_start` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_tp_start
(
    char          *netweave_node,
    NWDS_HANDLE   *tp_handle,
    NWDS_SIZE      max_name_length,
    char          *transaction_name,
    NWDS_ITEM_LIST *control_items,
    NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
netweave_node	✓		A section name in the INI file that specifies the parameters for the communications layer between your application and the NetWeave agent. The name may be either a logical name or a node name.
tp_handle		✓	The handle to be used with subsequent function calls.
max_name_length	✓		The maximum number of bytes that can be copied to the <code>transaction_name</code> parameter.
transaction_name		✓	A printable representation of the <code>transaction_id</code> . It is terminated with a NULL byte.
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to a callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.



Related Functions

`nwds_tp_abort` on page 174.

`nwds_tp_commit` on page 175.

`nwds_tp_resume` on page 176.

`nwds_tp_status` on page 180.







NWDS_TP_STATUS

This function returns information about an active transaction, not about transactions that have already committed or aborted. The `nwds_tp_status` function is included in all NetWeave releases.

```
NWDS_ERRNO nwds_tp_status
(
    char          *netweave_node,
    char          *transaction_name,
    NWDS_ITEM_LIST *control_items,
    NWDS_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
netweave_node	✓		A section name in the INI file that specifies the parameters for the communications layer between your application and the NetWeave agent. The section name may be either a logical name or a node name.
transaction_name	✓		A printable representation of the transaction_id.
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to a callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_tp_abort` on page 174.

`nwds_tp_commit` on page 175.

`nwds_tp_resume` on page 176.

`nwds_tp_start` on page 177.



NWDS_TRIGGER_CANCEL

This function dissociates a callback function from an event on a particular file. The `nwds_trigger_cancel` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_trigger_cancel
(
    NWDS_HANDLE      file_handle,
    NWDS_ITEM_LIST    *control_items,
    NWDS_CALL_BACK    *call_back);
```

Parameter	Input	Output	Description
file_handle	✓		The handle from a call to <code>nwds_file_open</code> .
control_items	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to a callback structure. If NULL, the call is synchronous.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.

For more information about the return codes, see page 229.

Related Functions

`nwds_trigger_read` on page 184.

`nwds_trigger_register` on page 186.





NWDS_TRIGGER_READ

This function returns the old and/or new records associated with the following trigger events:

- Add: a copy of the new record is returned in the `new_buffer`.
- Delete: a copy of the deleted record is returned in the `old_buffer`.
- Update: a copy of the original record is in the `old_buffer` and a copy of the changed record is in the `new_buffer`.

NOTE: Use the register to determine whether you should be passing old and/or new buffers.

The `nwds_trigger_read` function is provided as part of NetWeave's Data Server option. A `nwds_trigger_read` call is always synchronous because NetWeave queues the record(s) for the application before calling the callback from `nwds_trigger_register`.

```
NWDS_ERRNO nwds_trigger_read
    NWDS_HANDLE      file_handle,
    NWDS_FILTER_CLASS *trigger_type,
    NWDS_SIZE         maximum_length_new_buffer,
    void              *new_buffer,
    NWDS_SIZE         *actual_length_new_buffer,
    NWDS_ROW_VERSION  *new_row_version,
    NWDS_SIZE         maximum_length_old_buffer,
    void              *old_buffer,
    NWDS_SIZE         *actual_length_old_buffer,
    NWDS_ROW_VERSION  *old_row_version,
    NWDS_TRANS_ID     *transaction_id,
    NWDS_ITEM_LIST     *control_items);
```

Parameter	Input	Output	Description
file_handle	✓		The handle returned from a call to <code>nwds_file_open</code> .
trigger_type		✓	The trigger type of the particular change to the file.
maximum_length_new_buffer	✓		The maximum number of bytes that may be copied to <code>new_buffer</code> . If the record exceeds this size, only <code>maximum_length_new_buffer</code> bytes are copied to the buffer and <code>NWDS_DATA_OVERFLOW</code> is returned.
new_buffer	✓		An array in the user's data space where NetWeave will put the new record image.
actual_length_new_buffer		✓	The number of bytes written to <code>new_buffer</code> .
new_row_version		✓	An identifier to pass on update to synchronize updates from multiple users.



Parameter	Input	Output	Description
maximum_length_old_buffer	✓		The maximum number of bytes that may be copied to old_buffer. If the record exceeds this size, only maximum_length_old_buffer bytes are copied to the buffer and NWDS_DATA_OVERFLOW is returned.
old_buffer	✓		An array in the user's data space where NetWeave will put the original record image.
actual_length_old_buffer		✓	The number of bytes written to old_buffer.
old_row_version		✓	An identifier to pass on update to synchronize updates from multiple users.
transaction_id	✓		The transaction name returned from a call to nwds_tp_start.
control_items	✓		A pointer to an array of system-specific parameters.

Return Code (output)

Return code	Description
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling nwds_init() and before calling nwds_exit(). Call nwds_init() and re-issue the NetWeave function call.
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.

For more information about the return codes, see page 229.

Related Functions

nwds_trigger_cancel on page 182.

nwds_trigger_register on page 186.





NWDS_TRIGGER_REGISTER

This function associates a callback function with an event (a combination of add, delete, update) on a particular file. An application process that wants to monitor changes to the file must open the file. Call `nwds_trigger_read` to read the data that was the basis for the trigger event. The `nwds_trigger_register` function is provided as part of NetWeave's Data Server option.

```
NWDS_ERRNO nwds_trigger_register
    (NWDS_HANDLE      file_handle,
     NWDS_FILTER_CLASS trigger_type,
     NWDS_ITEM_LIST    *item_list,
     NWDS_CALL_BACK    *call_back,
     NWDS_CALL_BACK    *data_received);
```

Parameter	Input	Output	Description
file_handle	✓		The identifier returned from a call to <code>nwds_file_open</code> .
trigger_type	✓		Mask of trigger types: <code>nwds_trigger_update</code> , <code>nwds_trigger_write</code> , and/or <code>nwds_trigger_delete</code> . For the complete list of trigger functions, see page 14 or <code>netweave.h</code> .
item_list	✓		A pointer to an array of system-specific parameters.
call_back	✓		A pointer to a callback structure. If NULL, the call is synchronous.
data_received	✓		A pointer to the callback structure containing the function to call when a trigger message is received. For example, this callback function might call <code>nwds_trigger_read</code> to retrieve the message.

Return Code (output)

Return code	Description
NWDS_BAD_HANDLE	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed.
NWDS_NOT_IMPLEMENTED	The function is not available on the current platform.
NWDS_NOT_INITIALIZED	All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_INVALID_OPERATION	The attempted operation is not appropriate to the target object.

For more information about the return codes, see page 229.



Related Functions

`nwds_trigger_cancel` on page 182.

`nwds_trigger_read` on page 184.





Item Types and Values

In `netweave.h`, constants are defined for item types and values. An item list is an array of structures that declare the parameters that control a remote function call or receive information from a remote function. The item list structure can accommodate both constant and variable length parameter values.

Let's look at the definition of a single element of an item list called `nwds_item_list` using the C `sizeof` operator for numeric values:

```
typedef struct {  
    NWDS_ITEM_TYPE    type;  
    NWDS_SIZE         length;  
    void              *item;  
} NWDS_ITEM_LIST;
```

Parameter	Description
item type	Identifies the parameter that is being supplied in this element.
item_length	The number of bytes in the item_buffer that constitute the value of the parameter.
item	A pointer to a memory location where the value of the parameter is stored.

Some common item list definitions appear on the next page.



Common Item List Definitions

Assigning a Constant Length Value to a Parameter

```
long          RBA; /*RBA holds the relative byte address for UNIX flat file.*/
NWDS_ITEM_LIST I_list [2];

/*Load the item list to pass to nwds_file_position.*/
I_list[ 0 ] . type = NWDS_CFILE_FTELL;
I_list[ 0 ] . length = sizeof( long );
I_list[ 0 ] . item = &RBA;

I_list[ 1 ] . type = NWDS_END_OF_LIST; /*terminates the list-required*/
```

Assigning a Variable Length Value to a Parameter

```
short          primary_key = 0; /*Enscribe primary keyspecifier*/
short          exact_mode = NWDS_TAN_EXACT_POS;
/*primary_key_value will be set to the key value to match*/
char           primary_key_value [30];

NWDS_ITEM_LIST I_list [4];

/*Prepare the item list to pass to nwds_file_position.*/
I_list[ 0 ] . type = NWDS_TAN_KEYID;
I_list[ 0 ] . length = ( NWDS_SIZE )sizeof( short );
I_list[ 0 ] . item = &primary_key;

I_list[ 1 ] . type = NWDS_TAN_MODE;
I_list[ 1 ] . length = ( NWDS_SIZE )sizeof( short );
I_list[ 1 ] . item = &exact_mode;

I_list[ 2 ] . type = NWDS_TAN_KEYVALUE;
/*I_list[ 2 ] . length-set by the program*/
I_list[ 2 ] . item = primary_key_value;

I_list[ 3 ] . type = NWDS_END_OF_LIST; /*terminates the list - required*/
```





Message Queue (FIFO) Files

The FIFO (for **F**irst-**I**n, **F**irst-**O**ut) messaging queue provides a simple and robust interface for store-and-forward message delivery in a heterogeneous computing environment. A process that writes to a FIFO is called a producer; a process that reads from a FIFO is called a consumer. Messages written to a FIFO are appended to the end (tail) of the queue, while messages read from a FIFO are taken from the beginning (head) of the queue.

A FIFO queue is implemented as a ring of segments. You specify the size and number of segments when you create the FIFO, and each segment is stored as a record in a special file that NetWeave maintains. The local file system determines the maximum segment size (i.e. the maximum record size that is supported by the file system). One message may span multiple segments. The maximum message size is `nwds_max_user_size`.

When you set up the FIFO queue, make it local to a producer and remote to a consumer. This way, even if the communications layer fails, the producer is not interrupted. When communications are restored, the consumer(s) may resume processing the queue.

There are two ways to read messages from the queue:

- Using a single call to read the message and advance the head pointer. This gives better throughput and is intended for applications where more than one consumer process reads messages from the queue.
- Using two calls: the first to read the message, and the second (`nwds_file_position`) to advance the pointer. The two-call method (also called transaction mode) supports transaction processing because you can continue to reread a message until the head pointer is advanced to the next one.

For applications that read a queue by calling `nwds_file_read` asynchronously, NetWeave allows the read to complete either immediately with the error `NWDS_EOF`, or whenever a producer adds the next message to the queue. Because the read is asynchronous, the program is not blocked while waiting for its next message. Also, because NetWeave completes the read operation when the next message is added, the program does not poll the queue when it is empty.

Use the following functions for the message queue option:

Function	Description
<code>nwds_file_close</code>	Closes a message queue.
<code>nwds_file_create</code>	Creates a message queue.
<code>nwds_file_info</code>	Retrieves information about a queue.
<code>nwds_file_open</code>	Opens a message queue.
<code>nwds_file_position</code>	Completes a transaction involving a queue.
<code>nwds_file_read</code>	Reads the first message from the queue.
<code>nwds_file_remove</code>	Purges a message queue.
<code>nwds_file_write</code>	Appends a message to the end of the queue.



Generic C Files

A C file is a stream of bytes without any structure or indexing. Most platforms implement flat file structures that may be accessed from the standard I/O library of C.





NetWeave Kernel Functions for Windows NT

A NetWeave kernel function provides access to those proprietary features of the NT operating system that applications must share with NetWeave. *NT is the only Microsoft Windows operating system for which kernel functions are provided.*

You can use kernel functions to integrate asynchronous applications with NetWeave's API. Systems applications that need to wait on an event must use the NetWeave kernel functions to define both the object, and the callback function to associate with the object.

To implement fully asynchronous operation, NetWeave uses the NT systems call `WaitForMultipleObjects` and what are called NT-waitable objects. (In the NT environment, you can use waitable objects to signal a change of state – typically, to notify an application that an OS function call has completed.) Because NetWeave also uses waitable objects to synchronize its activities, there is a potential conflict between the NetWeave library and any applications that use NetWeave function calls. The kernel routines provide a way for the application to tell NetWeave about additional objects that it needs to wait for.

This section discusses the following kernel functions:

- `nwds_nt_clear_event`
- `nwds_nt_define_event`





NWDS_NT_CLEAR_EVENT

This function deregisters an object from the list of application objects on which NetWeave has to wait.

```
NWDS_ERRNO nws_nt_clear_event (  
    long          hEvent );
```

Parameter	Input	Output	Description
hEvent	✓		The object (Win32 handle) to be removed from the list on which NetWeave waits.

Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	The object was removed.
NWDS_PROCESS_NOT_FOUND	Invalid or unknown hEvent.





NWDS_NT_DEFINE_EVENT

The `nwds_nt_define_event` function specifies which application objects NetWeave must wait for. The callback function is an application function that NetWeave will call when the defined event is notified.

```
NWDS_ERRNO nwds_nt_define_event (  
    long          hEvent,  
    NWDS_CALL_BACK *completion);
```

Parameter	Input	Output	Description
hEvent	✓		The Win32 handle to monitor.
completion	✓		A pointer to the callback structure. This pointer cannot be NULL.

Return Code (output)

Return code	Description
NWDS_SUCCESSFUL	Registration was completed successfully.
NWDS_NO_MEMORY	Fatal error: insufficient heap space (the system is overloaded).
NWDS_DUPLICATE_PROCESS	The Win32 handle is already registered.



NetWeave Kernel Functions for UNIX

A NetWeave kernel function provides access to some proprietary feature of the operating system that an application must share with NetWeave. This section describes the NetWeave kernel functions for UNIX systems including Solaris, HPUX, AIX, Linux and DECUNIX TRU64.

You can use kernel functions to integrate asynchronous applications with NetWeave's API. NetWeave uses the *select()* function in the socket's library to wait for events to occur asynchronously. You may add your own socket to the list of sockets on which NetWeave waits. Because NetWeave must control the waiting process, any systems applications that need to wait on a socket event must use the NetWeave kernel functions to do so.

This section discusses the following kernel functions:

- `nwds_ux_clear_event`
- `nwds_ux_define_event`





NWDS_UX_CLEAR_EVENT

The `nwds_ux_clear_event` function removes a socket descriptor from the list of application file descriptors that NetWeave is waiting for.

```
NWDS_ERRNO nwds_ux_clear_event (  
    int      fileDescriptor,  
    int      readMask,  
    int      writeMask);
```

Parameter	Input	Output	Description
fileDescriptor	✓		The descriptor to remove from the list.
readMask	✓		Set to True or False to match the original setting when the mask was defined in <code>nwds_ux_define_event</code> .
writeMask	✓		Set to True or False to match the original setting when the mask was defined in <code>nwds_ux_define_event</code> .

Return Code (output)

Return code	Description
NWDS_BAD_PARAMETER	readMask or writeMask is not specified.
NWDS_SUCCESSFUL	The event flag was cleared.





NWDS_UX_DEFINE_EVENT

The `nwds_ux_define_event` function tells NetWeave which application sockets it has to wait for. The read callback function is an application function that NetWeave calls when the socket has data. The write callback function is an application function that NetWeave calls when the socket becomes writeable.

If the *permanent* parameter is TRUE, you must use `nwds_ux_clear_event` to remove the socket descriptor from the list for which NetWeave waits. If FALSE, NetWeave waits once and then removes the descriptor automatically.

```
NWDS_ERRNO nwds_ux_define_event (  
    int                fileDescriptor,  
    int                permanent,  
    NWDS_CALL_BACK    *readCompletion,  
    NWDS_CALL_BACK    *writeCompletion);
```

Parameter	Input	Output	Description
fileDescriptor	✓		The file descriptor that NetWeave must monitor.
permanent	✓		Set to TRUE if you intend to remove this event by a call to <code>nwds_ux_clear_event</code> .
readCompletion	✓		A pointer to the read callback structure. It cannot be NULL.
writeCompletion	✓		A pointer to the write callback structure. It cannot be NULL.

Return Code (output)

Return code	Description
NWDS_BAD_PARAMETER	One of the completion structures is invalid.
NWDS_NO_MEMORY	Fatal error: insufficient heap space.



NetWeave Kernel Functions for DEC, VMS, and OpenVMS

A NetWeave kernel function provides access to those proprietary features of the operating system that applications must share with NetWeave. This section describes the NetWeave kernel functions for VMS and OpenVMS.

You can use kernel functions to integrate asynchronous applications with NetWeave's API. NetWeave uses Asynchronous System Traps (ASTs) to notify a kernel layer that read or write has completed on a communications channel. The NetWeave kernel layer uses event flags to coordinate and synchronize service calls made on behalf of remote applications. Because NetWeave must control the waiting process, any systems applications that need to wait on an event must use the NetWeave kernel functions to do so.

In the DEC environment, you can use waitable objects to signal a change of state – typically, to notify an application that an OS function call has completed. Because NetWeave also uses event flags to synchronize its activities, there is a potential conflict between the NetWeave library and any applications that use NetWeave function calls. The kernel routines provide a mechanism that the application can use to tell NetWeave about additional event flags that it needs to wait for.

This section discusses the following kernel functions:

- `nwds_vms_clear_event`
- `nwds_vms_define_event`





NWDS_VMS_CLEAR_EVENT

The `nwds_vms_clear_event` function deregisters an event flag from the list of application event flags that NetWeave is waiting for.

```
NWDS_ERRNO nwds_vms_clear_event (  
    int          event_flag);
```

Parameter	Input	Output	Description
event_flag	✓		The event flag number to be cleared.

Return Code (output)

Return code	Description
NWDS_PROCESS_NOT_FOUND	Invalid or unknown event flag number.
NWDS_SUCCESSFUL	The event flag was cleared.





NWDS_VMS_DEFINE_EVENT

The `nwds_vms_define_event` function tells NetWeave which application event flags it has to wait for. The callback function is an application function that NetWeave calls after an event flag has been set. NetWeave does not set or clear an application event flag.

```
NWDS_ERRNO nwds_vms_define_event (  
    int          event_flag,  
    NWDS_CALL_BACK *completion);
```

Parameter	Input	Output	Description
event_flag	✓		The event flag number that NetWeave must monitor.
completion	✓		A pointer to the callback structure. It cannot be NULL.

Return Code (output)

Return code	Description
NWDS_DUPLICATE_PROCESS	The event flag is already set.
NWDS_NO_MEMORY	Fatal error: insufficient heap space.
NWDS_SUCCESSFUL	The event flag was set.



NetWeave Kernel Functions For Tandem

A NetWeave kernel function provides access to some proprietary feature of the operating system that an application must share with NetWeave. You can use kernel functions to integrate asynchronous applications with NetWeave's API. The Kernel Library provides the callback mechanism for the Tandem. Because the NetWeave function calls use the kernel library to implement the asynchronous API, you must use the kernel library for multithreaded asynchronous applications.

There are two distinct groups of functions in the kernel library for Tandem. One group manages asynchronous operations on files; the other manages asynchronous responses to system messages. Each group has a unique data structure and unique function prototype associated with it. First we describe the structure and prototype associated with asynchronous file I/O. Then we will review the structure and prototype for processing system messages.

This section discusses the following kernel functions:

- `nwds_kernel_call_back`
- `nwds_kernel_recv_call_back`
- `nwds_tandem_waitiox`
- `nwds_tandem_clear_events`
- `nwds_tandem_clear_system_events`
- `nwds_tandem_define_event`
- `nwds_tandem_define_system_event`
- `nwds_tandem_replyx`
- `nwds_tandem_receiveinfo`





NWDS_KERNEL_CALL_BACK

This structure has two elements that are analogous to the elements of the regular callback structure. `User_context` is a pointer to allocated memory where the application identifies the situation in which this file I/O event occurs. The callback procedure is unique to file I/O events.

```
Typedef struct {
    NWDS_CONTEXT          userContext,
    NWDS_KERNEL_CALL_BACK_PROC *procedure
} NWDS_KERNEL_CALL_BACK;
```

The procedure prototype for a `NWDS_KERNEL_CALL_BACK_PROC` looks like this:

```
typedef void (NWDS_KERNEL_CALL_BACK_PROC) (
    short          fileHandle,
    void           *buffer,
    short          length,
    short          guardianError,
    long           userTag,
    void           *userContext
);
```

Parameter	Input	Output	Description
fileHandle	✓		The Guardian file number where an I/O has just completed.
buffer	✓		The address of an area in the user's data space where the message is returned.
length	✓		The number of bytes in the message in the buffer.
guardianError	✓		The Guardian error code associated with the I/O. 0 means success; any other value indicates a problem.
userTag	✓		The tag specified when the I/O was initiated by a call to <code>NWDS_TANDEM_DEFINE_EVENT</code> .
userContext	✓		The same pointer as the structure's <code>userContext</code> parameter. Whatever you pass to NetWeave, NetWeave returns to you.





NWDS_KERNEL_RECV_CALL_BACK

Applications use this structure to tell NetWeave how to respond to system messages. The Tandem Guardian operating system posts special system messages to an application to alert it to unsolicited external events. The functions `NWDS_TANDEM_DEFINE_SYSTEM_EVENT` and `NWDS_TANDEM_CLEAR_SYSTEM_EVENTS` enable an application to tell NetWeave which system messages (events) it wishes to monitor and how to react to them.

```
typedef struct {
    NWDS_CONTEXT          userContext;
    NWDS_KERNEL_RECV_CALL_BACK_PROC *procedure;
} NWDS_KERNEL_RECV_CALL_BACK;
```

The `userContext` is a pointer to the application's memory space where the application identifies what is happening. The callback procedure contains parameters designed to return all the information that Guardian provides with the message.

```
typedef NWDS_MSG_USED (NWDS_KERNEL_RECV_CALL_BACK_PROC) (
    void          *buffer,
    short         length,
    short         guardianError,
    long          userTag,
    NWDS_RECEIVEINFO *info,
    void          *userContext
);
```

Parameter	Input	Output	Description
buffer	✓		The address of an area in the user's data space where the message is returned.
length	✓		The number of bytes in the message in the buffer.
guardianError	✓		The Guardian error code associated with the I/O. 0 means success; any other value indicates a problem.
userTag	✓		The tag specified when the I/O was initiated by a call to <code>NWDS_TANDEM_DEFINE_EVENT</code> .
info	✓		This structure contains the information obtained from Guardian, described below.
userContext	✓		The <code>userContext</code> you passed to NetWeave is returned to you.

The `NWDS_RECEIVEINFO` structure contains information obtained from Guardian on the application's behalf. NetWeave calls the Guardian procedure `FILE_GETRECEIVEINFO` to populate this structure. For detailed information about this structure, see the *Guardian Procedures Calls* manual. NetWeave



provides the function NWDS_TANDEM_RECEIVEINFO to retrieve this information about a user message. For more information about this function, see page 227.

```
typedef struct {  
    short      io_type;  
    short      max_reply_count;  
    short      msg_tag;  
    short      file_num;  
    short      sync_id[2];  
    short      sender_process_handle[10];  
    short      open_label;  
} NWDS_RECEIVEINFO;
```

Parameter	Description
io_type	Type of I/O caller.
max_reply_count	Maximum number of bytes for call to reply().
msg_tag	Tag to use in call to reply().
file_num	Sender's file number.
sync_id	The sync ID associated with this message.
sender_process_handle	Process handle of the process that sent the last message.
open_label	The value assigned by the application to the open connection on which the received message was sent.

The user's callback procedure must return a value that NetWeave interrogates to determine whether additional actions are required. The legitimate return codes for your procedures are:

NWDS_RECV_MSG_USED = 1

NWDS_RECV_MSG_NOT_USED = 2



NWDS_TANDEM_AWAITIOX

Use the NWDS_TANDEM_AWAITIOX function instead of the standard Guardian AWAITIOX call to complete an asynchronous I/O. (For COBOL users on Tandem, the function is named nwds_tand_awaitiox.)

```
NWDS_ERRNO NWDS_TANDEM_AWAITIOX (  
    short          *user_filenumbers,  
    void           *user_buffer,  
    short          *user_buffer_length,  
    long           *user_tag,  
    NWDS_RECEIVEINFO **receiveinfo,  
    short          *user_error,  
    long           user_timeout,  
    short          loop_forever);
```

Parameter	Input	Output	Description
user_filenumbers		✓	The Guardian file number of the I/O that just completed.
user_buffer		✓	The address of the area where the message is stored. Note for COBOL users: this parameter does not exist for the COBOL version of this call.
user_buffer_length		✓	The length of the message in the user_buffer.
user_tag		✓	The tag associated with the I/O when it was initiated.
receiveinfo		✓	The location of the NWDS_RECEIVEINFO structure that describes this request. For more information, see NWDS_KERNEL_RECV_CALL_BACK on page 214
user_error		✓	The Guardian error code from the I/O completion.
user_timeout	✓		The length of time (in hundredths of a second) to wait for the I/O to complete. If this value is negative, NetWeave waits forever.
loop_forever	✓		A logical that implements polling. If TRUE, the call is repeated with the same timeout value until an I/O completes.





NWDS_TANDEM_CLEAR_EVENTS

The `nwds_tandem_clear_events` function removes the callback(s) associated with either all callbacks, or a specified file number.

```
NWDS_ERRNO NWDS_TANDEM_CLEAR_EVENTS (  
    short      file_number,  
    long       tag  
    short      error);
```

Parameter	Input	Output	Description
file_number	✓		The Guardian file number returned from OPEN. If negative, all callbacks on all the user's files are cleared.
tag	✓		The tag specified when a particular I/O was initiated by a call to NWDS_TANDEM_DEFINE_EVENT. If negative, all callbacks for the file_number are cleared.
error	✓		A Guardian error code to pass to the callback(s).





NWDS_TANDEM_CLEAR_SYSTEM_EVENTS

This function removes the callback associated with a specified Guardian system error number. For more information, see `nwds_Tandem_define_system_event` on page 222.

```
NWDS_ERRNO NWDS_TANDEM_CLEAR_SYSTEM_EVENTS (  
    Short      system_message_number,  
    NWDS_KERNEL_RECV_CALL_BACK  *call_back);
```

Parameter	Input	Output	Description
system_message_number	✓		The Guardian system error number whose associated callback you want to remove. If this number is negative, all callbacks on all the user's system error codes are cleared.
call_back	✓		The pointer to the callback function that a call to <code>NWDS_TANDEM_DEFINE_SYSTEM_EVENT</code> originally associated with the system message code.





NWDS_TANDEM_DEFINE_EVENT

The `nwds_tandem_define_event` function associates an I/O operation on the specified file with a callback function and specific program context. When the I/O completes, the callback function is called with one argument being the program context.

An application that calls the NetWeave API must never call the Guardian procedure `awaitio(x)`. If your application must wait for I/O on a file unknown to NetWeave, you must call the NetWeave function `NWDS_TANDEM_AWAITIOX`.

```
NWDS_ERRNO NWDS_TANDEM_DEFINE_EVENT (
    short          file_number,
    long           tag,
    long           timeout,
    short          permanent,
    NWDS_ITEM_LIST *item_list,
    NWDS_KERNEL_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
file_number	✓		The Guardian file number returned from OPEN.
tag	✓		The tag specified when the I/O was initiated by a call to NWDS_TANDEM_DEFINE_EVENT.
timeout	✓		The length of time in centi-seconds to wait for the I/O to complete. If timeout is negative, wait is forever.
permanent	✓		A logical that indicates whether the callback is associated with only the next I/O or all subsequent I/O on this file number. Two constants are provided for this parameter: <ul style="list-style-type: none">• NWDS_NOT_PERMANENT (default): the callback applies only to the next I/O• NWDS_PERMANENT: the callback is associated with all I/Os
item_list	✓		A pointer to an array of system-specific parameters. No item types are presently defined for the kernel library.
call_back	✓		A pointer to the kernel callback structure. Unlike the standard API, this cannot be NULL.





NWDS_TANDEM_DEFINE_SYSTEM_EVENT

The `nwds_tandem_define_system_event` function provides notification of system messages. This function call must include the system message number and the callback function pointer, and indicate whether this registration is permanent.

```
NWDS_ERRNO NWDS_TANDEM_DEFINE_SYSTEM_EVENT (  
    short                system_message_number,  
    short                permanent,  
    NWDS_ITEM_LIST      *item_list,  
    NWDS_KERNEL_RECV_CALL_BACK *call_back);
```

Parameter	Input	Output	Description
system_message_number	✓		A Guardian system number.
permanent	✓		A logical that indicates whether the callback is associated with only the next I/O or all subsequent I/O on this system message number. Two constants are provided for this parameter: <ul style="list-style-type: none">• <code>NWDS_NOT_PERMANENT</code>: the callback applies only to the next I/O• <code>NWDS_PERMANENT</code> (default): the callback is associated with all I/Os
item_list	✓		A pointer to an array of system-specific parameters. No item types are presently defined for the kernel library.
call_back	✓		A pointer to a kernel receive callback structure. This cannot be NULL.





NWDS_TANDEM_REPLYX

The `nwds_tandem_replyx` function, which mimics the behavior of Guardian's `replyx` system call, gives Pathway server programmers additional flexibility and an alternative to `nwds_ipc_write` for returning replies. To use `nwds_tandem_replyx`, you should retrieve at least two parameters (`max_reply_count` and `msg_tag`) from the `nwds_receiveinfo` structure associated with the original request message.

For more information, see `nwds_tandem_receiveinfo` on page 227.

```
typedef struct {
    short    io_type;          /*returns type of I/O caller used (write,
writeread, etc)*/
    short    max_reply_count; /*returns max # bytes for call to reply()*/
    short    msg_tag;          /*returns tag to use in call to reply()*/
    short    file_num;         /*returns sender's file num*/
    short    sync_id[2];       /*The sync ID associated with this message*/
    short    sender_process_handle[10];
    short    open_label;
} NWDS_RECEIVEINFO;
```

```
NWDS_ERRNO NWDS_TANDEM_REPLYX (
    short    tag,
    short    reply_length,
    void      *reply,
    short    error);
```

Parameter	Input	Output	Description
tag	✓		The message tag parameter (<code>msg_tag</code>) from <code>nwds_receiveinfo</code> .
reply_length	✓		The length in bytes of the reply message. This amount must not exceed the <code>max_reply_count</code> from <code>nwds_receiveinfo</code> .
reply	✓		The address of the location containing the reply.
error	✓		This parameter may contain a Guardian error code to return to the calling application.





NWDS_TANDEM_RECEIVEINFO

The `nwds_tandem_receiveinfo` function gives Pathway server programmers more information about request messages. This function mimics the behavior of Guardian's

`FILE_GETRECEIVEINFO_` system call and returns the following information:

```
typedef struct {
    short      io_type;
    short      max_reply_count;
    short      msg_tag;
    short      file_num;
    short      sync_id[2];
    short      sender_process_handle[10];
    short      open_label;
} NWDS_RECEIVEINFO;
```

```
NWDS_ERRNO NWDS_TANDEM_REPLYX (
    NWDS_HANDLE      handle,
    NWDS_RECEIVEINFO *receive_info);
```

Parameter	Input	Output	Description
handle	✓		The process handle returned from the call to <code>nwds_ipc_accept</code> .
receive_info	✓	✓	The location where the <code>receive_info</code> structure will be stored.



Return Codes and Recovery

The table below lists the NetWeave return codes provided by the function `nwds_error_text` and gives suggestions for recovering from common problems. Please keep in mind the following:

- Where recovery is listed as “None,” a system fault has occurred that an application would not or could not normally address. If the error is associated with a connection or a file, the associated entity should be closed and re-opened. If the error occurs outside of a connection or file (for example, in a general purpose facility such as `nwds_init`), the application should seriously consider logging a message and terminating.
- Codes marked Discontinued are no longer used and should never occur.
- The designation *Exceptional user condition* means that an internal fault has occurred. Please contact NetWeave support and be ready to provide the error traces and INI file for the program that received the error.

Return code	It means	Suggested recovery
NWDS_ABORTED_BY_USER	Transaction is aborted by user	None.
NWDS_ACCESS_VIOLATION	Access violation	File security is enabled and the current user does not have access to the Guardian file. You should log in as a user who has the proper authority to view or update the file.
NWDS_ALREADY_EXISTS	File already exists	None. This condition is returned when someone tries to create a file or queue that already exists.
NWDS_BAD_ADDRESS	Invalid network address	Exceptional user condition. Please contact NetWeave support with the error traces and INI file for the program that received the error.
NWDS_BAD_HANDLE	Invalid NetWeave handle	You are trying to reuse a handle that has become invalid, usually because a file or connection was closed. Usually indicates a coding mistake by the user.
NWDS_BAD_INI_PARAMETER	Invalid/missing INI file parameter	See the error log to determine which INI file parameter is improper. Check and update your INI file, then retry the application.
NWDS_BAD_PARAMETER	Bad parameter passed	You are trying to call a function but one of your parameters is out of range. Usually indicates a coding mistake by the user. For more information, see the error log.
NWDS_BAD_PROCESS_NAME	Bad process name	Indicates a coding mistake by a Alpha/Open VMS user. You are trying to start a process with a name that is already in use. Stop the older version of the program.



Return code	It means	Suggested recovery
NWDS_BAD_PROTOCOL	Invalid network protocol	You are trying to make a connection over a communications protocol that your library does not support. Check for errors in your INI file.
NWDS_BAD_SERVER_NAME	Invalid NetWeave server name	This is usually a coding error that results from passing a name that is either empty or too long. For more information, see the error log
NWDS_CANNOT_REGION_LOCK	Discontinued	
NWDS_CI2_NOT_LOCKED		
NWDS_CI2_NOT_OUR_LOCK		
NWDS_CI2_SELF_LOCKED		
NWDS_DATA_OVERFLOW	Data overflow	More data is available than the user is prepared to receive. Review the specifications about the maximum messages you expect to send and receive, and adjust program parameters accordingly.
NWDS_DDL_INVALID_FIELD	Invalid DDL field	The INI file is invalid. One of the data types is not recognized.
NWDS_DDL_MISSING_FIELD	Missing DDL field	The INI file is invalid. An expected field is missing. Check and update your INI file.
NWDS_DDL_NOT_DEFINED	DDL_ENTRY not set for name	The INI file is incomplete or the message name does not match the group in the INI file that contains the metadata to describe the message. Check and update your INI file.
NWDS_DDL_SIZE_MISMATCH	DDL size does not match file	The data and data type are inconsistent.
NWDS_DELETE_FAILED	Delete failed	NWDS_TANDEM_CLEAR_TIMER_EVENTS returns this code when it cannot locate the event. If this occurs in well-tested procedures, it is an exceptional user condition.
NWDS_DELETED_RECORD	Discontinued	
NWDS_DIRECTORY_EXISTS		
NWDS_DISK_ERROR	Disk error received	None. For more information, see the error log.
NWDS_DLL_IN_USE	Discontinued	
NWDS_DLL_NOT_LOADED		



Return code	It means	Suggested recovery
NWDS_DUPLICATE_PROCESS	Duplicate process name	This runtime error is usually caused by an attempt to run an Agent or application server more than once.
NWDS_EOF	End of file	The user must evaluate the context of this condition to decide whether it is an expected or exceptional condition.
NWDS_EXECUTE_FAILED	Execute image failed	This code is specific to IBM/CICS. If it occurs during runtime, a long-running transaction may be running already.
NWDS_FILE_EXISTS	Discontinued	
NWDS_FILE_IN_USE	File is currently in use	A second user is trying to access a non-shared file.
NWDS_FILE_IS_FULL	File is full	None.
NWDS_FILE_MODIFIED_DURING_READS	Discontinued.	
NWDS_FILE_NOT_FOUND	File not found	The user must decide whether this is a problem or a non-fatal condition.
NWDS_FILE_NOT_OPEN	File has not been opened	You are trying to access a file that is not open. This usually indicates a coding mistake by the user. For more information, see the error log.
NWDS_ILLEGAL_FILENAME	Illegal file specification	The name is either blank, too long, or improper syntax for the remote system.
NWDS_INDEX_NOT_FOUND	Discontinued	
NWDS_INI_FILE_NOT_FOUND		
NWDS_INI_FILE_NOT_OPEN		
NWDS_INIT_ERROR		
NWDS_INVALID_DUPLICATE_KEY		
NWDS_INVALID_FILE_TYPE	Invalid file type	The file is not a FIFO. For more information, see the error log.
NWDS_INVALID_IO_OPERATION	Invalid I/O operation	This error is specific to Tandem Guardian. The user is trying to perform an action that is not supported for the target file type.
NWDS_INVALID_ITEM	Invalid item	An item in an item list is not of the proper data type or the value is out of range.
NWDS_INVALID_KEYED_RELATION	Discontinued	



Return code	It means	Suggested recovery
NWDS_INVALID_OPERATION	Invalid operation	The attempted operation is not appropriate to the target object. For example, trying to write to a file opened for read access. For more information, see the error log.
NWDS_INVALID_RECORD_NUMBER	Invalid record number	This code is specific to Alpha/Open VMS and means that the record number is out of range.
NWDS_INVALID_RECORD_SIZE	Discontinued	
NWDS_INVALID_SUBSTITUTION		
NWDS_INVALID_TRANSACTION_ID	Invalid transaction ID	The transaction is no longer active.
NWDS_IO_NOT_PROCESSED	Asynchronous I/O not processed	This condition is typical in the Tandem Guardian environment where a user application expects to receive messages about which NetWeave knows nothing.
NWDS_IO_PROCESSED	Asynchronous I/O processed	This is the normal completion code for calls to NWDS_TANDEM_AWAITIOX NetWeave has processed the message; retry the operation.
NWDS_ISCALL	Discontinued	
NWDS_ISHANGUP		
NWDS_ISNEWCLIENT		
NWDS_ITEM_INDEX		
NWDS_KERNEL_NOT_INITIALIZED	Kernel not initialized	This is an exceptional user condition. Please contact NetWeave support with the error traces and INI file for the program that received the error.
NWDS_KEY_MUST_BE_EXPLICIT	Discontinued	
NWDS_LIBRARY_ERROR	Library error	This is an exceptional user condition. Please contact NetWeave support with the error traces and INI file for the program that received the error.
NWDS_LINK_DOWN	Communication link is down	The user must assess whether this is a fatal or normal condition for the current circumstance.
NWDS_LOGON_DENIED	Logon denied	The user is not authorized to access the remote system.
NWDS_LOGON_DISABLED	Logon disabled	nwds_logon attempted but security is not enabled for this connection.
NWDS_LONG_RECORD	Discontinued	



Return code	It means	Suggested recovery
NWDS_MAX_SERVERS_RUNNING	Maximum servers running	This code is specific to Tandem SQL/MP. Increase the servers or try later.
NWDS_MUST_LOCK_IMPLICITLY	Discontinued	
NWDS_NAME_NOT_FOUND	Name not found in INI file	The name does not match any group in the INI file. Check and update your INI file.
NWDS_NO_DATA	No data available on handle	Assess whether this is a fatal or normal condition for this circumstance. If you did not expect an error, see the error log for more information.
NWDS_NO_MEMORY	Process out of memory	None. The system is overloaded.
NWDS_NO_NWDS_TRANS_ID	No Trans_ID supplied	The transaction is no longer active. For more information, see the error log.
NWDS_NO_OUTSTANDING_IO	No outstanding I/O	Usually indicates a coding problem: you have called <code>nwds_sleep</code> and there are no outstanding events. If this occurs in well-tested code, it is an exceptional user condition.
NWDS_NO_TRANSACTION	No transaction	The transaction is no longer active. For more information, see the error log.
NWDS_NO_TRANSACTION_MONITORING	No transaction monitoring	There is no TP monitor for this system.
NWDS_NOT_A_FILE	Discontinued	
NWDS_NOT_IMPLEMENTED	Feature not implemented	None. The function is not available on the current platform.
NWDS_NOT_INITIALIZED		All NetWeave functions must be called after calling <code>nwds_init()</code> and before calling <code>nwds_exit()</code> . Call <code>nwds_init()</code> and re-issue the NetWeave function call.
NWDS_NOT_ON_DIR	Discontinued	
NWDS_NOT_OPEN_FOR_DELETE	File not open for delete	This condition is specific for files on Alpha/Open VMS. Verify that the file is being opened properly for this type of operation.
NWDS_NOT_OPEN_FOR_UPDATE	File not open for update	
NWDS_NOT_OPEN_FOR_WRITE	File not open for write	For FIFOs, you are trying to write to a FIFO using a handle that is limited to reading messages.
NWDS_NOT_OPEN_WITH_INDEX	File not open with indexing	This condition is specific for files on Alpha/Open VMS. Verify that the file is being opened properly for indexed operations.



Return code	It means	Suggested recovery
NWDS_NOTHREAD	No thread active or supplied	This is an exceptional user condition. Please contact NetWeave support with the error traces and INI file for the program that received the error.
NWDS_OPEN_FAILED	Open file failed	The error log may contain additional information about why you cannot open this file. Check the INI file to verify that the name is spelled correctly and that the path has not been changed.
NWDS_OPERATION_FAILED	Operation failed	Usually indicates a coding mistake by the user. If this occurs in a well-tested program, treat it as an exceptional user condition. For more information, see the error log.
NWDS_PASSWORD_REQUIRED	Password is required after logon	Supply the password by calling <code>nwds_password</code> .
NWDS_PENDING		The operation has been initiated successfully. Final status and data will be delivered to the specified callback function.
NWDS_PORT_ALREADY_ATTACHED	Discontinued	
NWDS_PORT_NOT_ATTACHED		
NWDS_POSITION_FAILED	Position failed	This is specific to Tandem TM/MP when replicating structured files to a Guardian platform. It is an exceptional user condition.
NWDS_POSITION_UNDEFINED	Discontinued	
NWDS_PROCESS_NOT_CONNECTED	Process not connected	<code>nwds_session_close</code> returns this condition if the session is already closed.
NWDS_PROCESS_NOT_FOUND	Process not found	The error log should explain what happened. If it doesn't, treat this as an exceptional user condition.
NWDS_READ_FAILED	Read from file failed	
NWDS_RECORD_IN_USE	Record in use	
NWDS_RECORD_IS_LOCKED	Record is already locked	The attempt to lock a file record failed because the record is already locked.



Return code	It means	Suggested recovery
NWDS_RECORD_NOT_FOUND	Discontinued	
NWDS_RECORD_TOO_LONG		
NWDS_RECORD_TOO_SHORT		
NWDS_REJECTED		
NWDS_REQUEST_DENIED	Request denied by server	The user is not authorized to perform the function. For more information, see the error log.
NWDS_RESUBMIT_TRANSACTION	Discontinued	
NWDS_RMS_CLOSE_FAILED	RMS close failed	This condition is specific for files on Alpha/Open VMS. Obtain the VMS/RMS error code from the NetWeave error log and contact NetWeave Technical Support for a problem diagnosis.
NWDS_RMS_CONNECT_FAILED	RMS connect failed	
NWDS_RMS_CREATE_FAILED	RMS create failed	
NWDS_RMS_DELETE_FAILED	RMS delete failed	
NWDS_RMS_ERASE_FAILED	RMS erase failed	
NWDS_RMS_NODE_NAME	Discontinued	
NWDS_RMS_SEARCHLIST		
NWDS_RMS_UPDATE_FAILED	RMS update failed	This condition is specific for files on Alpha/Open VMS. Get the VMS/RMS error code from the NetWeave error log, and contact NetWeave Technical Support to diagnose the problem.
NWDS_RMS_WILDCARDS	Wildcard error	This condition is specific for files on Alpha/Open VMS and indicates that a name with wildcard characters is not permitted in this context.
NWDS_SECURITY_VIOLATION	Security violation	This user's access privileges do not allow this action. For more information, see the error log.
NWDS_SEEK_FAILED	Seek to position in file failed	The record does not exist or the position parameters are out of range for this system.
NWDS_SQL_BAD_PASSWORD	Discontinued	
NWDS_SQL_BAD_USERNAME		



Return code	It means	Suggested recovery
NWDS_SQL_ERROR	Error from SQL Server	Many different conditions can cause this code. For more information, see the error log.
NWDS_SQL_INVALID_COLUMN	Invalid SQL column index	The column name does not occur in this table.
NWDS_SQL_INVALID_VERB	Invalid SQL verb for call	The SQL statement is improper.
NWDS_SQL_LOGON_FAILED	Discontinued	
NWDS_STALE_ROW		
NWDS_STOP_FAILED	Stop process failed	This code is specific to Alpha/Open VMS. Stop the process from the system console.
NWDS_SYSTEM_NOT_FOUND	System node not recognized	For IBM/CICS: the monitor has restarted the router or the TCP/IP helper process. Any remote connections must be reestablished. For Tandem, Guardian error 18 was returned on a system call, and the connection must be restarted/reestablished.
NWDS_TAPE_PAST_EOT	Discontinued	
NWDS_TIMEOUT	Operation timed out	An operation timed out.
NWDS_TP_NOT_INITIALIZED	No transaction monitor supported	None. The TP monitor is not operating.
NWDS_TRANSACTION_ABORTED	Transaction was aborted	None. This is an informational message.
NWDS_TRANSACTION_ACTIVE	Transaction is active	
NWDS_TRANSACTION_COMMITTED	Discontinued	
NWDS_TRANSACTION_INCOMPLETE	Transaction incomplete	None. This is an informational message.
NWDS_UPDATE_FAILED	Update failed	Exceptional user condition. Please contact NetWeave support with the error traces and INI file for the program that received the error.
NWDS_UX_USER_DATA_TO_PROCESS	Discontinued	



Return code	It means	Suggested recovery
NWDS_VMS_CREPRC_FAILED	VMS SYS\$CREPRC failed	This condition is specific for processes on Alpha/Open VMS. The VMS System service CREPRC (Create Process) or GETJPI (Get Job Info) failed. See the NetWeave error log for more information, and contact NetWeave technical support for clarification.
NWDS_VMS_GETJPI_FAILED	VMS SYS\$GETJPI failed	
NWDS_WOULD_BLOCK	The system is overloaded and the call would block now	Because the system cannot complete the operation at this time, retry the call later.
NWDS_WRITE_FAILED	Write to file failed	This is usually a fatal condition. For more information, see the error log.
NWDS_WRONG_VERSION	Wrong version	Discontinued
NWDS_ZERO_DATA_SIZE	Attempt to write zero data	You cannot specify a message length of zero.



Return Code Numeric Definitions

This section details the NetWeave status and error code definitions and their numeric values. The Error code definitions are listed in alphabetical order and numeric order for ease of reference.

NWDS Status Codes (Non-Error)

NWDS_SUCCESSFUL	-1
NWDS_PENDING	-2
NWDS_EOF	-3
NWDS_ISCALL	-4
NWDS_ISHANGUP	-5
NWDS_REJECTED	-6
NWDS_ISNEWCLIENT	-7
NWDS_TRANSACTION_ACTIVE	-8
NWDS_TRANSACTION_COMMITTED	-9
NWDS_TRANSACTION_ABORTED	-10
NWDS_PASSWORD_REQUIRED	-11

NWDS Error Codes by Name

NWDS_ABORTED_BY_USER	234
NWDS_ACCESS_VIOLATION	114
NWDS_ALREADY_EXISTS	102
NWDS_BAD_ADDRESS	190
NWDS_BAD_HANDLE	189
NWDS_BAD_INI_PARAMETER	232
NWDS_BAD_PARAMETER	110
NWDS_BAD_PROCESS_NAME	149
NWDS_BAD_PROTOCOL	191
NWDS_BAD_SERVER_NAME	193
NWDS_CANNOT_REGION_LOCK	185
NWDS_CI2_NOT_LOCKED	204
NWDS_CI2_NOT_OUR_LOCK	205
NWDS_CI2_SELF_LOCKED	203
NWDS_DATA_OVERFLOW	121
NWDS_DDL_INVALID_FIELD	217



NWDS_DDL_MISSING_FIELD	216
NWDS_DDL_NOT_DEFINED	219
NWDS_DDL_SIZE_MISMATCH	218
NWDS_DELETE_FAILED	168
NWDS_DELETED_RECORD	161
NWDS_DIRECTORY_EXISTS	179
NWDS_DISK_ERROR	112
NWDS_DLL_IN_USE	230
NWDS_DLL_NOT_LOADED	229
NWDS_DUPLICATE_PROCESS	148
NWDS_EXECUTE_FAILED	147
NWDS_FILE_EXISTS	180
NWDS_FILE_IN_USE	104
NWDS_FILE_IS_FULL	111
NWDS_FILE_MODIFIED_DURING_READS	187
NWDS_FILE_NOT_FOUND	103
NWDS_FILE_NOT_OPEN	107
NWDS_ILLEGAL_FILENAME	105
NWDS_INDEX_NOT_FOUND	162
NWDS_INI_FILE_NOT_FOUND	200
NWDS_INI_FILE_NOT_OPEN	199
NWDS_INIT_ERROR	151
NWDS_INVALID_DUPLICATE_KEY	172
NWDS_INVALID_FILE_TYPE	181
NWDS_INVALID_IO_OPERATION	178
NWDS_INVALID_ITEM	206
NWDS_INVALID_KEYED_RELATION	165
NWDS_INVALID_OPERATION	122
NWDS_INVALID_RECORD_NUMBER	160
NWDS_INVALID_RECORD_SIZE	170
NWDS_INVALID_SUBSTITUTION	201
NWDS_INVALID_TRANSACTION_ID	209
NWDS_IO_NOT_PROCESSED	197
NWDS_IO_PROCESSED	196



NWDS_ITEM_INDEX	163
NWDS_KERNEL_NOT_INITIALIZED	207
NWDS_KEY_MUST_BE_EXPLICIT	164
NWDS_LIBRARY_ERROR	120
NWDS_LINK_DOWN	117
NWDS_LOGON_DENIED	214
NWDS_LOGON_DISABLED	213
NWDS_LONG_RECORD	175
NWDS_MAX_SERVERS_RUNNING	123
NWDS_MUST_LOCK_IMPLICITLY	184
NWDS_NAME_NOT_FOUND	220
NWDS_NO_DATA	195
NWDS_NO_MEMORY	152
NWDS_NO_NWDS_TRANS_ID	233
NWDS_NO_OUTSTANDING_IO	198
NWDS_NO_TRANSACTION	208
NWDS_NO_TRANSACTION_MONITORING	210
NWDS_NOT_A_FILE	182
NWDS_NOT_IMPLEMENTED	118
NWDS_NOT_INITIALIZED	239
NWDS_NOT_ON_DIR	186
NWDS_NOT_OPEN_FOR_DELETE	131
NWDS_NOT_OPEN_FOR_UPDATE	130
NWDS_NOT_OPEN_FOR_WRITE	133
NWDS_NOT_OPEN_WITH_INDEX	132
NWDS_NOTHREAD	238
NWDS_OPEN_FAILED	143
NWDS_OPERATION_FAILED	113
NWDS_PORT_ALREADY_ATTACHED	188
NWDS_PORT_NOT_ATTACHED	177
NWDS_POSITION_FAILED	167
NWDS_POSITION_UNDEFINED	202
NWDS_PROCESS_NOT_CONNECTED	119
NWDS_PROCESS_NOT_FOUND	106



NWDS_READ_FAILED	144
NWDS_RECORD_IN_USE	173
NWDS_RECORD_IS_LOCKED	116
NWDS_RECORD_NOT_FOUND	166
NWDS_RECORD_TOO_LONG	171
NWDS_RECORD_TOO_SHORT	169
NWDS_REQUEST_DENIED	215
NWDS_RESUBMIT_TRANSACTION	231
NWDS_RMS_CLOSE_FAILED	142
NWDS_RMS_CONNECT_FAILED	141
NWDS_RMS_CREATE_FAILED	140
NWDS_RMS_DELETE_FAILED	135
NWDS_RMS_ERASE_FAILED	136
NWDS_RMS_NODE_NAME	137
NWDS_RMS_SEARCHLIST	138
NWDS_RMS_UPDATE_FAILED	134
NWDS_RMS_WILDCARDS	139
NWDS_SECURITY_VIOLATION	115
NWDS_SEEK_FAILED	146
NWDS_SQL_BAD_PASSWORD	225
NWDS_SQL_BAD_USERNAME	224
NWDS_SQL_ERROR	223
NWDS_SQL_INVALID_COLUMN	227
NWDS_SQL_INVALID_VERB	228
NWDS_SQL_LOGON_FAILED	226
NWDS_STALE_ROW	212
NWDS_STOP_FAILED	150
NWDS_SYSTEM_NOT_FOUND	109
NWDS_TAPE_PAST_EOT	174
NWDS_TIMEOUT	194
NWDS_TP_NOT_INITIALIZED	235
NWDS_TRANSACTION_INCOMPLETE	211
NWDS_UPDATE_FAILED	176
NWDS_UX_USER_DATA_TO_PROCESS	236



NWDS_VMS_CREPRC_FAILED	222
NWDS_VMS_GETJPI_FAILED	221
NWDS_WOULD_BLOCK	237
NWDS_WRITE_FAILED	145
NWDS_WRONG_VERSION	183
NWDS_ZERO_DATA_SIZE	192

NWDS Error Codes by numeric value

102NWDS_ALREADY_EXISTS
103NWDS_FILE_NOT_FOUND
104NWDS_FILE_IN_USE
105NWDS_ILLEGAL_FILENAME
106NWDS_PROCESS_NOT_FOUND
107NWDS_FILE_NOT_OPEN
109NWDS_SYSTEM_NOT_FOUND
110NWDS_BAD_PARAMETER
111NWDS_FILE_IS_FULL
112NWDS_DISK_ERROR
113NWDS_OPERATION_FAILED
114NWDS_ACCESS_VIOLATION
115NWDS_SECURITY_VIOLATION
116NWDS_RECORD_IS_LOCKED
117NWDS_LINK_DOWN
118NWDS_NOT_IMPLEMENTED
119NWDS_PROCESS_NOT_CONNECTED
120NWDS_LIBRARY_ERROR
121NWDS_DATA_OVERFLOW
122NWDS_INVALID_OPERATION
123NWDS_MAX_SERVERS_RUNNING
130NWDS_NOT_OPEN_FOR_UPDATE
131NWDS_NOT_OPEN_FOR_DELETE
132NWDS_NOT_OPEN_WITH_INDEX
133NWDS_NOT_OPEN_FOR_WRITE
134NWDS_RMS_UPDATE_FAILED
135NWDS_RMS_DELETE_FAILED



136NWDS_RMS_ERASE_FAILED
137NWDS_RMS_NODE_NAME
138NWDS_RMS_SEARCHLIST
139NWDS_RMS_WILDCARDS
140NWDS_RMS_CREATE_FAILED
141NWDS_RMS_CONNECT_FAILED
142NWDS_RMS_CLOSE_FAILED
143NWDS_OPEN_FAILED
144NWDS_READ_FAILED
145NWDS_WRITE_FAILED
146NWDS_SEEK_FAILED
147NWDS_EXECUTE_FAILED
148NWDS_DUPLICATE_PROCESS
149NWDS_BAD_PROCESS_NAME
150NWDS_STOP_FAILED
151NWDS_INIT_ERROR
152NWDS_NO_MEMORY
160NWDS_INVALID_RECORD_NUMBER
161NWDS_DELETED_RECORD
162NWDS_INDEX_NOT_FOUND
163NWDS_ITEM_INDEX
164NWDS_KEY_MUST_BE_EXPLICIT
165NWDS_INVALID_KEYED_RELATION
166NWDS_RECORD_NOT_FOUND
167NWDS_POSITION_FAILED
168NWDS_DELETE_FAILED
169NWDS_RECORD_TOO_SHORT
170NWDS_INVALID_RECORD_SIZE
171NWDS_RECORD_TOO_LONG
172NWDS_INVALID_DUPLICATE_KEY
173NWDS_RECORD_IN_USE
174NWDS_TAPE_PAST_EOT
175NWDS_LONG_RECORD
176NWDS_UPDATE_FAILED



177NWDS_PORT_NOT_ATTACHED
178NWDS_INVALID_IO_OPERATION
179NWDS_DIRECTORY_EXISTS
180NWDS_FILE_EXISTS
181NWDS_INVALID_FILE_TYPE
182NWDS_NOT_A_FILE
183NWDS_WRONG_VERSION
184NWDS_MUST_LOCK_IMPLICITLY
185NWDS_CANNOT_REGION_LOCK
186NWDS_NOT_ON_DIR
187NWDS_FILE_MODIFIED_DURING_READS
188NWDS_PORT_ALREADY_ATTACHED
189NWDS_BAD_HANDLE
190NWDS_BAD_ADDRESS
191NWDS_BAD_PROTOCOL
192NWDS_ZERO_DATA_SIZE
193NWDS_BAD_SERVER_NAME
194NWDS_TIMEOUT
195NWDS_NO_DATA
196NWDS_IO_PROCESSED
197NWDS_IO_NOT_PROCESSED
198NWDS_NO_OUTSTANDING_IO
199NWDS_INI_FILE_NOT_OPEN
200NWDS_INI_FILE_NOT_FOUND
201NWDS_INVALID_SUBSTITUTION
202NWDS_POSITION_UNDEFINED
203NWDS_CI2_SELF_LOCKED
204NWDS_CI2_NOT_LOCKED
205NWDS_CI2_NOT_OUR_LOCK
206NWDS_INVALID_ITEM
207NWDS_KERNEL_NOT_INITIALIZED
208NWDS_NO_TRANSACTION
209NWDS_INVALID_TRANSACTION_ID
210NWDS_NO_TRANSACTION_MONITORING



211NWDS_TRANSACTION_INCOMPLETE
212NWDS_STALE_ROW
213NWDS_LOGON_DISABLED
214NWDS_LOGON_DENIED
215NWDS_REQUEST_DENIED
216NWDS_DDL_MISSING_FIELD
217NWDS_DDL_INVALID_FIELD
218NWDS_DDL_SIZE_MISMATCH
219NWDS_DDL_NOT_DEFINED
220NWDS_NAME_NOT_FOUND
221NWDS_VMS_GETJPI_FAILED
222NWDS_VMS_CREPRC_FAILED
223NWDS_SQL_ERROR
224NWDS_SQL_BAD_USERNAME
225NWDS_SQL_BAD_PASSWORD
226NWDS_SQL_LOGON_FAILED
227NWDS_SQL_INVALID_COLUMN
228NWDS_SQL_INVALID_VERB
229NWDS_DLL_NOT_LOADED
230NWDS_DLL_IN_USE
231NWDS_RESUBMIT_TRANSACTION
232NWDS_BAD_INI_PARAMETER
233NWDS_NO_NWDS_TRANS_ID
234NWDS_ABORTED_BY_USER
235NWDS_TP_NOT_INITIALIZED
236NWDS_UX_USER_DATA_TO_PROCESS
237NWDS_WOULD_BLOCK
238NWDS_NOTHREAD
239NWDS_NOT_INITIALIZED



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.