Raw DDE

Sanford A. Staab

Created: March 20, 1992

ABSTRACT

Dynamic data exchange (DDE) is one of the least understood capabilities of the Microsoft® Windows™ graphical environment. For the most part, this lack of understanding results from the absence of a highly detailed explanation of the correct protocol to use when performing DDE transactions. This article describes each type of DDE transaction in a table format that outlines what operations the client and server applications must perform to complete the transaction properly.

The DDEML.DLL library in Windows version 3.1 removes the need for most applications to deal with raw DDE transactions. However, understanding proper DDE transaction protocol is a necessity when creating or communicating with applications that do not use DDEML.

INTRODUCTION

In this article, each possible dynamic data exchange (DDE) transaction is represented by a table. The client column of the table shows the actions taken by the client application. The server column of the table shows the actions taken by the server application. Each row represents a point in time with time elapsing with subsequent rows.

Before presenting all possible legal DDE transaction sequences, it is necessary to introduce some abbreviated syntax, in the form of functions, to clarify and unclutter the transaction descriptions.

Post Actions

All posted DDE messages are similar in that the *wParam* parameter always holds the window handle from which the message was posted. The *lParam* parameter, however, varies from message to message and usually contains two values. In versions 3.0 and 3.1 of the Microsoft® Windows™ graphical environment, these values are kept in the **LOWORD** and **HIWORD** of *lParam.* In the Win32™ Application Programming Interface, these parts are packed within a structure pointed to by *lParam.* When an application posts a DDE message, the table will display:

Post(*msg, lo, hi, status, format*)

- *msg* is the WM_DDE message being posted. The *msg* parameter is described using the distinguishing letters of the message name and may optionally have a bracketed qualifier explaining the context of the message. The message type and context dictate the *lo* and *hi* values.
- *lo* and *hi* are the respective parts of *lParam* that distinguish the message.
- *status* and *format* are optional pieces of information that are packed into the global data handle being passed within *lParam*.

status flags pertaining to the transaction sequence are listed. A '!' preceding a flag name means that it must be FALSE; if the '!' is absent, the flag must be TRUE. A '?' indicates that flag settings do not matter in a transaction. A parameter set to '–' is not applicable.

For example:

Post(**ACK**[**execute**], ***!fAck***, ***hCommands***, **–**, **–**)

means that a WM_DDE_ACK message was posted in response to a WM_DDE_EXECUTE message. The **LOWORD** of *lParam* had the *fAck* bit clear. The **HIWORD** of *lParam* had the *hCommands* data handle. This data handle does not contain any extra information, so two dashes indicate that this information is not applicable.

Table 1 lists all DDE messages and contexts possible along with their corresponding *lo* and *hi* values.

Table 1. DDE Messages and Contexts

Message and context	**LOWORD of *lParam***	**HIWORD of *lParam***	**Information in data handle**	**Sent from**	**Comments**
INIT	*aApp*	*aItem*		Client	Sent only.
ACK[init]	*aApp*	*aItem*		Server	Sent only.
ACK[advise]	*wStatus*	*aItem*		Server	
ACK[data]	*wStatus*	*aItem*		Client	
ACK[exec]	*wStatus*	*hCommands*		Server	
ACK[hotdata]	*wStatus*	*aItem*		Client	
ACK[poke]	*wStatus*	*aItem*		Server	
ACK[request]	*wStatus*	*aItem*		Server	

ACK[unadvise]	*wStatus*	*aItem*		Server	
ADVISE	*hOptions*	*aItem*	*wStatus, format*	Client	The *fRelease* bit is ignored and always assumed to be TRUE.
DATA[request]	*hData*	*aItem*	*wStatus, format*	Server	*fRequest* is always TRUE.
DATA[hot]	*hData*	*aItem*	*wStatus, format*	Server	*fRequest* is always FALSE.
DATA[warm]	0	*aItem*		Server	*fAck* is assumed FALSE.
EXECUTE	0	*hCommands*		Client	An ACK should always be generated regardless of the *fAck* bit

					value.
POKE	*hData*	*aItem*	*wStatus, format*	Client	An ACK should always be generated regardless of the *fAck* bit value.
REQUEST	*format*	*aItem*		Client	
TERMINATE	0	0		Client or Server	Spontaneously generated or in response to a TERMINATE.
UNADVISE	*format*	*aItem*		Client	

The *wStatus* parameter in Table 1 may contain any of the flags in Table 2.

Table 2. *wStatus* Flag Values

Flag	Notes
fRelease	Indicates that the receiver of the data handle is to free the data handle memory. POKE, DATA, and ADVISE

	transactions responding with a negative ACK require the sender to free the data handle instead.
fAckReq	Indicates that the receiver must post an ACK message in response. Some messages imply this.
fAck	This is the same value as the *fAckReq* flag and is set in ACK messages. When set, the ACK message is a positive ACK. When cleared, the message is a negative ACK (NACK for short).
fDeferUpd	Used only in an ADVISE message to indicate whether the link will be hot or warm. When set, the link is warm.
Other values	Any other values in the *wStatus* word should be ignored.

Send Actions

A syntax similar to **Post** is used for sending messages during the initiate sequences.

Receive Events

A **Receive**(*msg,* [*flags*]) implies that the current context has received the message in question. This indicates that this receive event triggered the actions that follow. For clarity, we may include the optional *flags* parameter to show key status flags that distinguish the message.

Create Actions

A **Create**(*ObjectType*) describes the actions an application takes to create the object, including standard allocation and initialization of the object. In the case of DDE data handles, this involves calling the **GlobalAlloc** function. In the case of atoms, this involves calling the **GlobalAddAtom** function. Note that creation could have happened at the time indicated by the table or previously. This action also includes the creation or copying of any data implied by the data within the data handle. For example, this would include the creation of a bitmap for CF_BITMAP data.

Reuse Actions

A **Reuse**(*ObjectType*) action indicates that the application can reuse the object given to it by a preceding **Post** or that it may have freed the object and later recreated it.

Free Actions

Generally the **Free**(*ObjectType*) action is the inverse of the **Create**(*ObjectType*) action. In the case of data handles, this involves calling the **GlobalFree** function. In the case of atoms, this involves calling the **GlobalDeleteAtom** function. **Free** actions can happen at the time noted or later. This includes the freeing of any indirect data implied by the object's contents.

TRANSACTION TABLES

DDE is always initiated by the client application sending or broadcasting (via **SendMessage**) a WM_DDE_INITIATE message. When a server application receives this message, it checks the application and topic atoms to decide whether it should respond. Should it decide to do so, it sends back a WM_DDE_ACK message to the client, thus establishing a DDE connection. Table 3 outlines this action.

Table 3. Initiation

Client	Server	Comments
Create(*aAppClient*) **Create**(*aTopicClient*) **Send**(INIT, *aAppClient*, *aTopicClient*)		The client sends or broadcasts a WM_DDE_INITIATE message to all potential servers. *aAppClient* and *aTopicClient* may be 0 to indicate a wild initiate. All top-level windows are

		potential servers.
	Receive(INIT) **Create**(*aAppServer*) **Create**(*aTopicServer*) **Send**(ACK[init], *aAppServer*, *aTopicServer*)	When a potential server wants to respond to the client's offer, it posts a WM_DDE_ACK message back to the client, letting the client know the server's window handle.
Receive(ACK[init], *aAppServer*, *aTopicServer*)) **Free**(*aAppServer*) **Free**(*aTopicServer*) . . . After **Send**(INIT) is compl		Once the client receives the ACK message, both windows are locked into a DDE conversation. A WM_DDE_TERMINATE must be posted from either the client or the server to close the conversation properly.

eted: **Free**(*aAppClient*) **Free**(*aTopicClient*)		

Table 4. REQUEST—Server Sets *fRelease*

Client	Server	Comments
Create(*aItemClient*) **Post**(REQUEST, *format, aItemClient*, –, –)		The client application posts a WM_DDE_REQUEST message to the server, asking for data that *aItemClient* references and in the format specified.
	Receive(REQUEST) **Reuse**(*aItemClient*) **Create**(*hDataServer*) **Post**(DATA[request], *hDataServe*	The server receives the REQUEST message and decides to post a data message containing the requested data. The server sets the *fRelease* bit, which tells the client that it is

Client	Server	Comments
	r, aItem Client, fRequest | *fRelease, format*)	responsible for freeing the data. Because the *fAck* bit is not set, the client should not ACK the data message and therefore must accept responsibility for freeing the data. The *fRequest* bit indicates that this data message is in response to a REQUEST message.
Receive(DATA[request]) **Free**(*hDataClient*) **Free**(*aItemClient*)		The client receives the data and must eventually free the data handle and atom.

Table 5. REQUEST—Server Sets *fAckReq*

Client	Server	Comments
Creat		The client

e(*aItemClient*) **Post**(REQUEST, *format, aItemClient*, –, –)		application posts a WM_DDE_REQUEST message to the server, asking for data that *aItemClient* references and in the format specified.
	Receive(REQUEST) **Reuse**(*aItemClient*) **Create**(*hDataServer*) **Post**(DATA[request], *hDataServer, aItemClient, fRequest* | *fAckReq* | *! fRelease, format*)	The server receives the REQUEST message and decides to post a data message containing the requested data. The server clears the *fRelease* bit, which tells the client that it is not responsible for freeing the data. Because the *fAck* bit is set, the client should ACK the data message. The *fRequest* bit indicates that this data message is in

		response to a REQUEST message.
Receive(DATA[request]) **Reuse**(*aItemClient*) **Post**(ACK[data], ?, *aItemClient*, –, –)		The client receives the data and must post an ACK or a NACK because the *fAck* bit was set in the data message.
	Receive(ACK[data]) **Free**(*aItemClient*) **Free**(*hDataServer*)	The server receives the data ACK and is responsible for freeing the data handle and atom.

Table 6. REQUEST—Server Sets *fRelease* and *fAckRequest*—Client ACKs

Client	Server	Comments
Create(aItemClient) **Post**(REQUEST, *format, aItem*		The client application posts a WM_DDE_REQUEST message to the server, asking for data that *aItemClien*

Client, –, –)		*t* references and in the format specified.
	Receive(REQUEST) **Reuse**(*aItemClient*) **Create**(*hDataServer*) **Post**(DATA[request], *hDataServer, aItemClient, fRequest* | *fRelease* | *fAckReq, format*)	The server receives the REQUEST message and decides to post a data message containing the requested data. The server sets the *fRelease* bit, which tells the client that it is responsible for freeing the data. Because the *fAck* bit is set, the client should ACK the data message. The *fRequest* bit indicates that this data message is in response to a REQUEST message.
Receive(DATA[request])		The client receives the data and decides to

Free(*hDataClient*) **Reuse**(*aItemClient*) **Post**(ACK[data], *fAck, aItemClient*, –, –)		post an ACK. This tells the server and the system that the client has accepted responsibility for freeing the data handle.
	Receive(ACK[data], *fAck*) **Free**(*aItemClient*)	The server receives the data ACK and therefore is not responsible for freeing the data handle, only the atom.

Table 7. REQUEST—Server Sets *fRelease* and *fAckRequest*—Client NACKs

Client	Server	Comments
Create(*aItemClient*) **Post**(REQUEST, *format, aItemClient*, –, –)		The client application posts a WM_DDE_REQUEST message to the server, asking for data that *aItemClient* references and in the format specified.

	Receive(REQUEST) **Reuse**(*aItemClient*) **Create**(*hDataServer*) **Post**(DATA[request], *hDataServer, aItemClient, fRequest* | *fRelease* | *fAckReq, format*)	The server receives the REQUEST message and decides to post a data message containing the requested data. The server sets the *fRelease* bit, which tells the client that it is responsible for freeing the data. Because the *fAck* bit is set, the client should ACK the data message. The *fRequest* bit indicates that this data message is in response to a REQUEST message.
Receive(DATA[request]) **Free**(*hDataClient*)		The client receives the data and decides to post a NACK. This tells the server that it has responsibili

Client	Server	Comments
Reuse(*altemClient*)		ty for freeing the data handle.
Post(ACK[data], *!fAck, aItemClient*, –, –)		
	Receive(ACK[data], *!fAck*) **Free**(*hDataServer*) **Free**(*aItemClient*)	The server receives the data ACK and is responsible for freeing the data handle because the *fAck* bit is clear. The server should then free the data handle and the atom.

Table 8. REQUEST—Server NACKs

Client	Server	Comments
Create(*altemClient*) **Post**(REQUEST, *format, aItemClient*, –, –)		The client application posts a WM_DDE_REQUEST message to the server, asking for data that *aItemClient* references and in the format specified.

	Receive(REQUEST) **Reuse**(*aItemClient*) **Post**(ACK[request], *!fAck, aItemClient*)	The server receives the REQUEST message and decides to post a negative ACK message, which informs the client that the data is not available in the format requested.
Receive(ACK[request]) **Free**(*aItemClient*)		The client receives the NACK message, completing the transaction.

Table 9. POKE—Client Clears *fRelease*

Client	Server	Comments
Create(*aItemClient*) **Create**(*hDataClient*) **Post**(POKE, *hDataClient*, *aItemClient*, *!*		The client posts a POKE message containing the data, item, and format information. The *fRelease* bit is clear, indicating that the client retains responsibility for

fRelease, format)		freeing the data handle. Note that the *fAck* bit is not used. POKE messages always imply *fAck* = TRUE.
	Receive(POKE) **Reuse**(*aItemClient*) **Post**(ACK[poke], ?, *aItemClient*)	The server receives the POKE message and must post an ACK message in response. Because the *fRelease* bit is clear, the server must not free the data handle memory.
Receive(ACK[poke]) **Free**(*aItemClient*) **Free**(*hDataClient*)		The client receives the ACK message and, regardless of the *fAck bit,* must free the data handle.

Table 10. POKE—Client Sets *fRelease*—Server ACKs

Client	Server	Comments

Create(*aItemClient*) **Create**(*hDataClient*) **Post**(POKE, *hDataClient*, *aItemClient*, *fRelease, format*)		The client posts a POKE message containing the data, item, and format information. The *fRelease* bit is set, indicating that the server should free the data handle if it positively ACKs the data. Note that the *fAck* bit is not used. POKE messages always imply *fAck* = TRUE.
	Receive(POKE) **Reuse**(*aItemClient*) **Free**(*hDataServer* {*server*}) **Post**(ACK[poke], *fAck, aItemClient*)	The server receives the POKE message and must post an ACK message in response. Because the *fRelease* bit is set, the server must free the data handle memory.
Receive(A		The client receives

CK[poke], *fAck*) **Free**(*aItemClient*)		the ACK and frees the atom.

Table 11. POKE—Client Sets *fRelease*—Server NACKs

Client	Server	Comments
Create(*aItemClient*) **Create**(*hDataClient*) **Post**(POKE, *hDataClient*, *aItemClient*, *fRelease, format*)		The client posts a POKE message containing the data, item, and format information. The *fRelease* bit is set, indicating that the server should free the data handle if it positively ACKs the data. Note that the *fAck* bit is not used. POKE messages always imply *fAck* = TRUE.
	Receive(POKE) **Reuse**(*aItemClient*) **Post**(ACK[	The server receives the POKE message and must post a NACK message in response.

	poke], *!fAck, aItemClient*)	The client must free the data handle memory because of the negative ACK.
Receive(ACK[poke], *!fAck*) **Free**(*hDataClient* {*client* }) **Free**(*aItemClient*)		The client receives the ACK and frees the atom and data handle because the ACK was negative.

Table 12. EXECUTE

Client	Server	Comments
Create(*hCommands*) **Post**(EXECUTE, 0, *hCommands*, –, –)		The client posts an EXECUTE message that contains raw text for execution. This data handle contains flags.
	Receive(EXECUTE) **Post**(ACK[exec], ?,	The server receives the EXECUTE and posts an execute ACK, which should

Client	Server	Comments
	hCommands{*server*})	contain the same data handle that was given to it in the EXECUTE.
Receive(ACK[exec]) **Free**(*hCommands*{*client*})		The client receives the ACK[exec] message and frees the data handle. The status flags show the client whether the execute was successful.

Table 13. ADVISE—Server ACKs

Client	Server	Comments
Create(*aItemClient*) **Create**(*hOptions*{*client*}) **Post**(ADVISE, *hOptions, aItemClient*, ?, *format*)		The client posts an ADVISE message with the item and format desired with which to be linked. The flags within the *hOptions* data handle indicate whether the link is hot or warm (*fDeferUpd*) and whether the server

Client	Server	Comments
		is allowed to outrun the client (*fAck*).
	Receive(ADVISE) **Reuse**(*alternClient*) **Free**(*hOptions*{*server*}) **Post**(ACK[advise], *fAck, altemClient*)	The server receives the ADVISE message and returns a positive ACK to the client. This makes the server responsible for freeing the data handle.
Receive(ACK[advise], *fAck*) **Free**(*altemClient*)		The client receives the positive ACK and thus does not need to free the data handle. It then frees the atom.

Table 14. ADVISE—Server NACKs

Client	Server	Comments
Create(*alternClient*) **Create**(*hOptions*		The client posts an ADVISE message with the item and format desired

{*client*}) **Post**(ADVISE, *hOptions, aItemClient*)		with which to be linked. The flags within the *hOptions* data handle define whether the link is hot or warm (*fDeferUpd*) and whether the server is allowed to outrun the client (*fAck*).
	Receive(ADVISE) **Reuse**(*aItemClient*) **Post**(ACK[advise], *! fAck, aItemClient*)	The server receives the ADVISE message and returns a negative ACK to the client. This makes the client responsible for freeing the data handle.
Receive(ACK[advise], *! fAck*) **Free**(*hOptions*{*client*}) **Free**(*aItemClient*)		The client receives the negative ACK and thus must free the data handle. It then frees the atom.

Table 15. UNADVISE

Client	Server	Comments
Create(*atemClient*) **Post**(UNADVISE, *format*, *aItemClient*)		The client posts an UNADVISE message, which indicates the format and item of the link it wants to close.
	Receive(UNADVISE) **Reuse**(*atemClient*) **Post**(ACK[unadvise], ?, *aItemClient*)	The server receives the UNADVISE message and posts a positive or negative ACK back to the client.
Receive(ACK[unadvise]) **Free**(*aItemClient*)		The client receives the ACK[unadvise] and frees the associated atom.

Table 16. ADVISE DATA—Warm Link

Client	Server	Comments

	Create(*aItemServer*) **Post**(DATA[warm], 0, *aItemServer*)	The server posts a warm link DATA message to inform the client that the data associated with the atom specified has changes.
Receive(DATA[warm]) **Free**(*aItemServer*)		The client receives the DATA[warm] message and frees the associated atom.

Table 17. ADVISE DATA—Hot Link without *fAck*

Client	Server	Comments
	Create(*aItemServer*) **Post**(DATA[hot], *hDataServer, aItemServer, ! fAck* | *fRelease, format*)	The server posts a hot link DATA message to pass the new data to the client. The *fRelease* bit gives the client responsibility for freeing the data.
Receive(D		The client receives

Client	Server	Comments
ATA[request]) **Free**(*hDataClient*{*client*}) **Free**(*aItemServer*)		the DATA[hot] message and frees the associated atom and data.

Table 18. ADVISE DATA—Hot Link with *fAck*—Server Clears *fRelease*

Client	Server	Comments
	Create(*aItemServer*) **Post**(DATA[hot], *hDataServer, aItemServer, fAck* | *! fRelease, format*)	The server posts a hot link DATA message to pass the new data to the client. The *!fRelease* bit lets the server keep responsibility for freeing the data.
Receive(DATA[hot]) **Reuse**(*aItemServer*) **Post**(ACK[data], ?, *aItem*		The client receives the DATA[hot] message and posts an ACK message to the server. Note that, because the *fRelease* bit was

Client, –, –)		clear, the *fAck* state of the ACK message has no effect on who frees the data handle.
	Receive(ACK[data]) **Free**(*aItemServer*) **Free**(*hDataServer*{*server*})	The server receives the ACK message and, regardless of the *fAck* state, must free its data handle eventually.

Table 19. ADVISE DATA—Hot Link with *fAck*—Server Sets *fRelease*—Client ACKs

Client	Server	Comments
	Create(*aItemServer*) **Post**(DATA[hot], *hDataServer, aItemServer, fAck* | *fRelease, format*)	The server posts a hot link DATA message to pass the new data to the client. The *fRelease* bit gives the client responsibility for freeing the data.
Receive(DATA[h		The client receives the

ot]) **Reuse**(*aItemServer*) **Free**(*hDataClient*{*client*}) **Post**(ACK[data], *fAck, aItemClient*, –, –)		DATA[hot] message and frees the associated atom and data. It then posts an ACK message to indicate to the server that the DATA message was handled by the client.
	Receive(ACK[data], *fAck*) **Free**(*aItemServer*)	The server receives the ACK and only frees the atom because the client freed the data handle.

Table 20. ADVISE DATA—Hot Link with *fAck*—Server Sets *fRelease*—Client NACKs

Client	Server	Comments
	Create(*aItemServer*) **Post**(DATA[hot], *hDataServer, aItemServer, fAck* | *fRelease,*	The server posts a hot link DATA message to pass the new data to the client. The *fRelease* bit gives the client responsibility for freeing the data.

	format)	
Receive(DATA[hot]) **Reuse**(*aItemServer*) **Free**(*hDataClient* {*client*}) **Post**(ACK[data], *!fAck, aItemClient*, –, –)		The client receives the DATA[hot] message and frees the associated atom and data. It then posts a negative ACK message to indicate to the server that the client did not handle the DATA message.
	Receive(ACK[data], *fAck*) **Free**(*aItemServer*) **Free**(*hDataServer* {*server*})	The server receives the negative ACK and frees the atom and data handle on the server side.

TERMINATE Transactions

Either the client or the server application may initiate TERMINATE transactions. When an application posts a WM_DDE_TERMINATE message, the DDE protocol calls for that application not to post any further DDE messages. If the application should receive any DDE messages other than the responding WM_DDE_TERMINATE message, the protocol states that the application should free any objects associated with the message.

This is not quite correct. If a WM_DDE_DATA message is posted to an application that does not have the *fRelease* bit set, the receiver should not free this data because the data may have been posted to several other applications as well.