DDE

DDE Overview
Server and Client
DDE Conversations
J Commands and Events
Communications Protocol
Examples

DDE Overview

Dynamic Data Exchange allows Windows programs to communicate with each other.

DDE allows two Windows programs to exchange data by posting messages to each other using a standard protocol. J provides a comprehensive set of DDE commands so that you can communicate between J and any other Windows program supporting DDE, including another copy of J. You may have up to 20 DDE conversations at a time.

For example, with DDE you could set up J as a calculation server, so that another Windows program could send it a sentence for evaluation and receive back the result.

The DDE interface uses the Window Driver. Messages are *sent* with Window Driver commands, for example:

```
   wd 'ddereq jserver calc res'
```

Messages are *received* as a Windows event, and require handlers, named `sys_`*`eventname`*, for each event.

For example, here is a typical result of `wd'q'`, following a ddepoke event:

```
   wdq
+----------+-----------+
|syshandler|sys_handler|
+----------+-----------+
|sysevent  |sys_ddepoke|
+----------+-----------+
|sysdefault|sys_default|
+----------+-----------+
|systopic  |top        |
+----------+-----------+
|sysitem   |item       |
+----------+-----------+
|sysdata   |+/\i.10    |
+----------+-----------+
```

To respond to a `ddepoke`, you define a handler named `sys_ddepoke`, for example the following is used in file system\packages\dde\server2.ijs:

```
sys_ddepoke=: 3 : 0
sysdata=: sysdata -. CRLF,TAB
write0 sysdata
if. sysdata -: 'exit' do. return. end.
if. sysdata -: 'off' do.
  delay 1
  signoff ''
end.
try. val=: ".sysdata
catch. val=: 'unable to execute: ',sysdata end.
if. FORMAT do. val=: ":val end.
write1 val
topitem=: topitem,systopic;sysitem;<val
```

)

Most Windows applications support DDE in some form or another. However, not all applications support the complete set of DDE commands, and in some cases the command usage is not standard. With J, you write programs to handle the DDE interface, and therefore you can tailor the J side of the protocol to fit the other application's requirements.

In this chapter we discuss general principles of DDE with examples of a J to J DDE link, and then discuss a link between J and *Microsoft Word for Windows*. Several other examples, including a link to *Microsoft Visual Basic* will be found in the directory: system\examples\dde.

Server and Client

The two parties to a DDE conversation (*connection* or *link*) are known as the *server* and *client*. The server makes itself available by registering itself with Windows. The client initiates the conversation by sending a message to the server.

The DDE protocol is asymmetric: the client initiates the conversation, sends messages and instructions to the server, and terminates the conversation. The server cannot by itself initiate a conversation, or send messages except in response to a client request.

DDE Conversations

There are three types of DDE conversations, known as *hot*, *warm* and *cold*:

- in a hot link, the client asks that whenever some data changes, the server send it the new data.

- in a warm link, the client asks that whenever some data changes, the server send it an indication that the data has changed (but not the new data itself).

- in a cold link, the server never notifies the client whenever data has changed.

  Programs to handle each of these will be slightly different, and you will need to know what type of conversation the other application supports.

  Hot and warm links are also known as *advise loops*.

  A warm link is essentially a combination of a hot and cold link - the hot link advises of a change in the data item, the cold link is used to send the data item.

A conversation uses three identifiers: *service*, *topic* and *item*. A conversation may also move *data:*

- **service** identifies the server application, and is typically set by the server
- **topic** is a high level identifier of the information being exchanged
- **item** is a low level identifier of the information being exchanged
- **data** is the data being exchanged (as character data)

Service, topic, and item names are case-insensitive.

Some applications require specific topic and item names be used; others do not care. A commonly used topic is the *system* topic, which may also have a *sysitems* item, both are used to provide information about a server application to a client. For example, Lotus 123 for Windows has a system topic:

```
   wd 'ddereq 123w system sysitems'
SysItems Topics Formats RangeNames Selection Status

   wd 'ddereq 123w system status'
Ready
```

Some applications such as spreadsheets allow data to be exchanged in clipboard format. The utility `clipfmt` in script format.ijs will convert a list or table into clipboard format.

With spreadsheets, it is typical for the topic to be the worksheet name, and the item to be the cell or range name. For example, suppose `data` is a table of shape two by three, then this could be written to an Excel worksheet `sheet1` as follows:

```
   txt=. clipfmt data
   wd 'ddepoke excel sheet1 r1c1:r2c3 *',txt
```

Some servers support a DDE *execute* command, allowing the client to use the server's command language directly. However, servers that support execute usually do so only in the system topic, and then only in a limited fashion, for example to run menu commands such as File/Open. For non-trivial tasks it is usually best to create a macro in the application and then use the execute command to run the macro. Most applications follow the conventions for execute expressions indicated in the following:

```
[open("sample.xlm")]
[run("r1c1")]
[connect][download(query1,results.txt)][disconnect]
```

J Commands and Events

The following summarizes the Window Driver commands and event types. Here:

`S`	is the service name
`T`	is a topic
`I`	is an item
`D`	is some data

Server commands:

`ddename S`	set service name `S`
`ddereqd D`	provide data `D` to a ddereq event. It must be the first command given after a ddereq event is received.
`ddeadvise T I D`	provides data `D` to advise loops with given topic and item. It returns 0 if the loop is no longer active.

Server events:

`ddepoke`	data has been sent
`ddeex`	execute command string
`ddereq`	request for data
`ddestart`	request to start advise loop

Client commands:

`ddecons`	return conversation names (service and topic)
`ddedis [S [T]]`	disconnect
`ddeex S T D`	execute command in server
`ddepoke S T I D`	poke data to server
`ddereq S T I`	request data from server
`ddestart S T I`	start advise loop
`ddestop S T I`	top advise loop

Client events:

`result`	answer to ddereq query
`ddeadvise`	new data from advise loop

Communications Protocol

Opening the conversation:
The client always initiates the conversation by sending a message to the server. The server must be ready to respond. How the server does this depends on the application.

For J to act as a server, you must have first loaded J and set the service name, either from the command line or using the ddename command. You also need event handlers for the events you want to respond to.

Client commands:
The client can send four types of messages:

- a *ddepoke* message sends data to the server. The server does not respond. How the server treats this data depends on the application. If the server is a spreadsheet, the data may be written to the current worksheet; if the server is J, it may be treated as a sentence to be executed.
- a *ddeex* message sends a command to the server. The server does not respond. How the server treats this depends on the application, but it would typically be used to invoke a macro command in the server. A J server does not have to recognise this message, since a ddepoke message can be used instead.
- a *ddereq* message sends a request for a data item to the server. The server responds with the data item.
- a *ddestart* message sets up an advise loop, asking the server to advise the client whenever a specific data item has changed. The *ddestop* message terminates the advise loop. Once an advise loop has been initiated, the client must be prepared to accept messages from the server at any time.

Closing the conversation:
A conversation is closed whenever either application terminates. A client can also close the conversation by issuing a *ddedis* command (received by Windows, but not sent to the server).

Examples

The following examples illustrate. First we set up a J to J connection manually:

Load two copies of J, and arrange the windows so that both are visible at the same time. You may find it helpful to minimize Program Manager to reduce screen clutter.

In one window (the server), define the service name as *jserver:*

```
   wd 'ddename jserver'
```

In the other window (the client), use the ddepoke command to send data to the server:

```
   wd 'ddepoke jserver abc xyz mydata'
```

In the server, check the value of `wdq`

```
   wdq
+----------+-----------+
|syshandler|sys_handler|
+----------+-----------+
|sysevent  |sys_ddepoke|
+----------+-----------+
|sysdefault|sys_default|
+----------+-----------+
|systopic  |abc        |
+----------+-----------+
|sysitem   |xyz        |
+----------+-----------+
|sysdata   |mydata     |
+----------+-----------+
```

This indicates the character string "`mydata`" was sent to the J server, with topic `abc` and item `xyz`.

Now terminate the connection from the client:

```
   wd 'ddedis'
```

In practice, you will want to set up a protocol that will enable client and server to communicate back and forth. To illustrate this we describe typical hot and cold links that use J as an execution server.

J to J Hot Link

The script system\packages\dde\server1.ijs implements the server side of a DDE hot link. The protocol is as follows:

- the server defines its service name to be: *jserver*
- the client issues a ddestart command, using any topic and item
- the client issues a ddepoke for this topic and item, with the data being an executable J sentence

- the server executes the sentence, and returns the result with a ddeadvise command
- the client can issue a ddepoke repeatedly. If the data is "`close`" the topic and item are closed. If the data is "`exit`" the J server program is exited. If the data is "`off`" the J server task is terminated.

If you have not already done so, load 2 copies of J and minimize Program Manager to reduce screen clutter. In one copy of J (the server), enter:

```
load 'system\packages\dde\server1.ijs'
```

If you wish, you could load one copy of J, and enter the following command to load the second copy as a server (change the directory reference as appropriate):

```
wd 'winexec "\j3\j.exe system\packages\dde\server1.ijs"'
```

You could also create a new Shortcut or Program Manager item, which loads J with this script file as its profile file.

The server displays a Windows dialog box, which will be used to illustrate the conversation. If you wish, you can also minimize the server window.

In the other copy of J (the client), enter the following to initialize the client

```
load'system\examples\dde\client1.ijs'
```

You should see some J code executed in the server window, and the result in the client session.

You can now send J sentences to be evaluated:

```
   cmd 'i.4 5'
 0  1  2  3  4
 5  6  7  8  9
10 11 12 13 14
15 16 17 18 19
```

`cmd` sends its argument using ddepoke:

```
cmd=: 3 : 0
wd 'ddepoke jserver top item *',y.
)
```

The server window shows the conversation.

Try other expressions, for example, to create and use names:

```
   cmd 'mydata=: 2 3 5 7'
2  3   5    7

   cmd '#mydata"'
4
```

To close the server, enter:

```
   cmd 'off'
```

(The server does not respond.)

J to J Cold Link

The script system\packages\dde\server2.ijs implements the server side of a DDE cold link. The protocol is as follows:

- the server defines its service name to be: *jserver*
- the client issues a ddepoke for any topic and item, with the data being an executable J sentence
- the server executes the sentence, and saves the result with the topic and item
- the client issues a ddereq command for the topic and item
- the server responds with the corresponding result
- the client can issue a ddepoke and ddereq repeatedly. If the ddepoke data is "`exit`" the J server program is exited. If the data is "`off`" the J server task is terminated.

As before, load 2 copies of J and minimize Program Manager. In the server copy of J, enter:

```
   load 'system\packages\dde\server2.ijs'
```

then minimize the server window.

To initialize the client copy of J:

```
   load'system\examples\dde\client1.ijs'
```

Use cmd to set expressions:

```
   cmd 'i.4 5'
0  1   2    3     4
5  6   7    8     9
10 11  12   13    14
15 16  17   18    19
```

`cmd` sends its argument using ddepoke, and retrieves the result using ddereq:

```
cmd=: 3 : 0
wd 'ddepoke jserver top item *',y.
wd 'ddereq jserver top item'
)
```

Try other expressions, then to close the server, send:

```
   cmd 'off'
```

(The server does not respond.)

Note that the hot link protocol is simpler than the cold link protocol, since once it is set up, the client need only issue one command to send a sentence and retrieve the result. Also, the ddereq

command used with the cold link may time out if the server is not ready to send the data item requested.