NSDPSContext

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	AppKit/NSDPSContext.h

Class Description

The NSDPSContext class is the programmatic interface to objects that represent Display PostScript System *contexts*. A context can be thought of as a *destination* to which PostScript code is sent for execution. Each Display PostScript context contains its own complete PostScript environment including its own local VM (PostScript Virtual Memory). Every context has its own set of stacks, including an operand stack, graphics state stack, dictionary stack, and execution stack. Every context also contains a **FontDirectory** which is local to that context, plus a **SharedFontDirectory** that is shared across all contexts. There are three built-in dictionaries in the dictionary stack. From top to bottom, they are **userdict**, **globaldict**, and **systemdict**. **userdict** is private to the context, while **globaldict** and **systemdict** are shared by all contexts. **globaldict** is a modifiable dictionary containing information common to all contexts. **systemdict** is a read-only dictionary containing all the PostScript operators.

At any time there is the notion of the *current context*. The current context for the current thread may be set using **setCurrentContext:.**

NSDPSContext objects by default write their output to a specified *data* destination. This is used for printing, faxing, and for generation of saved EPS (Encapsulated PostScript) code. The means to create contexts that interact with displays are platform-specific.

The NSApplication object creates an NSDPSContext by default.

NSDPSContext Objects and Display PostScript System Context Records

When an NSDPSContext object is created, it creates and manages a *DPSContext* record. Programmers familiar with the client side C function interface to the Display PostScript System can access the DPSContext record by sending a **context** message to an NSDPSContext object. You can then operate on this context record using any of the functions or single operator functions defined in the Display PostScript System client library. Conversely, you can create an NSDPSContext object from a DPSContext record with the **DPSContextObject()** function, as defined in “Client Library Functions”. You can then work with the created NSDPSContext object using any of the methods described here.

General Exception Conditions

A variety of exceptions can be raised from NSDPSContext. In most cases, exceptions are raised because of errors returned from the Display PostScript Server. Exceptions are listed under “Types and Constants.” Also see the *Display PostScript System, Client Library Reference Manual*, by Adobe Systems Incorporated, for more details on Display PostScript System error names and their possible causes.

Method Types

Initializing a context	– initWithMutableData:forDebuggin:languageEncoding: nameEncoding:textProc:errorProc:
Testing the drawing destination	– isDrawingToScreen
Accessing context data	– mutableData
Setting and identifying the current context	+ currentContext
	+ setCurrentContext:
	– DPSContext
Controlling the context	– flush
	– interruptExecution
	– notifyObjectWhenFinishedExecuting:
	– resetCommunications
	– wait
Managing returned text and errors	+ stringForDPSError:
	– errorProc
	– setErrorProc:
	– setTextProc:
	– textProc
Sending raw data	– printFormat:
	– printFormat:arguments:
	– writeData:
	– writePostScriptWithLanguageEncodingConversion:
Managing binary object sequences	– awaitReturnValues
	– writeBOSArray:count:ofType:
	– writeBOSNumString:length:ofType:scale:
	– writeBOSString:length:
	– writeBinaryObjectSequence:length:
	– updateNameMap

Managing chained contexts	– chainChildContext: – childContext – parentContext – unchainContext
Controlling the wait cursor	– startWaitCursorTimer – setWaitCursorEnabled: – isWaitCursorEnabled
Debugging aids	+ areAllContextsOutputTraced + areAllContextsSynchronized + setAllContextsOutputTraced: + setAllContextsSynchronized: – isOutputTraced – isSynchronized – setOutputTraced: – setSynchronized:

Class Methods

areAllContextsOutputTraced

+ (BOOL)**areAllContextsOutputTraced**

Returns YES if the data flowing between the application’s contexts and their destinations is copied to diagnostic output.

areAllContextsSynchronized

+ (BOOL)**areAllContextsSynchronized**

Returns YES if all NSPDSContext objects invoke the wait method after sending each batch of output.

currentContext

+ (NSDPSContext *)**currentContext**

Returns the current context of the current thread.

setAllContextsOutputTraced:

+ (void)**setAllContextsOutputTraced:**(BOOL)*flag*

Causes the data (PostScript code, return values, and so forth) flowing between the all the application's contexts and their destinations to be copied to diagnostic output.

setAllContextsSynchronized:

+ (void)**setAllContextsSynchronized:**(BOOL)*flag*

Causes the **wait** method to be invoked each time an NSDPSContext object sends a batch of output to its destination.

setCurrentContext:

+ (void)**setCurrentContext:**(NSDPSContext *)*context*

Installs *context* as the current context of the current thread.

stringForDPSError:

+ (NSString *)**stringForDPSError:**(const DPSBinObjSeqRec *)*error*

Returns a string representation of *error*.

Instance Methods

DPSContext

– (DPSContext)**DPSContext**

Returns the corresponding DPScontext.

awaitReturnValues

– (void)**awaitReturnValues**

Waits for all return values from the result table.

chainChildContext:

– (void)**chainChildContext:**(NSDPSContext *)*child*

Links *child* (and all of it's children) to the receiver as its chained context, a context that receives a copy of all PostScript code sent to the receiver.

childContext

– (NSDPSContext *)**childContext**

Returns the receiver's child context, or **nil** if none exists.

errorProc

– (DPSErrorProc)**errorProc**

Returns the context's error callback function.

flush

– (void)**flush**

Forces any buffered data to be sent to its destination.

initWithMutableData:forDebugging:languageEncoding:nameEncoding:textProc:errorProc:

– **initWithMutableData:**(NSMutableData *)*data* **forDebugging:**(BOOL)*debug*
languageEncoding:(DPSProgramEncoding)*langEnc*
nameEncoding:(DPSNameEncoding)*nameEnc* **textProc:**(DPSTextProc)*tProc*
errorProc:(DPSErrorProc)*errorProc*

Initializes a newly allocated NSDPSContext that writes itsoutput to *data* using the language and name encodingsspecified by *langEnc* and *nameEnc*. The callback functions *tProc* and *errorProc* handle text and errorsgenerated by the context. If *debug* is YES, the output is given in human-readable form in which large structures (such as images) may be represented by comments.

interruptExecution

– (void)**interruptExecution**

Interrupts execution in the receiver's context.

isDrawingToScreen

– (BOOL)**isDrawingToScreen**

Returns YES if the drawing destination is the screen.

isOutputTraced

– (BOOL)**isOutputTraced**

Returns YES if the data flowing between the application’s single context and its destination is copied to diagnostic output.

isSynchronized

– (BOOL)**isSynchronized**

Returns whether the **wait** method is invoked each time the receiver sends a batch of output to the server.

isWaitCursorEnabled

– (BOOL)**isWaitCursorEnabled**

Returns whether the wait cursor is enabled.

See also: **PScurrentwaitcursorenabled()**

mutableData

– (NSMutableData *)**mutableData**

Returns the receiver’s data object.

notifyObjectWhenFinishedExecuting:

– (void)**notifyObjectWhenFinishedExecuting:**(id)*object*

Registers *object* to receive a **contextFinishedExecuting**: message when the NSDPSContext’s destination is ready to receive more input.

parentContext

– (NSDPSContext *)**parentContext**

Returns the receiver’s parent context, or **nil** if none exists.

printFormat:

– (void)**printFormat:**(NSString *)*format*,...

Constructs a string from *format* and following string objects (in the manner of **printf()**) and sends it to the context’s destination.

printFormat:arguments:

– (void)**printFormat:**(NSString *)*format* **arguments:**(va_list)*argList*

Constructs a string from *format* and *argList* (in themanner of **vprintf()**) and sends it to the context’s destination.

resetCommunication

– (void)**resetCommunication**

Discards any data that hasn’t already been sent to its destination.

setErrorProc:

– (void)**setErrorProc:**(DPSErrorProc)*proc*

Sets the context’s error callback function to *proc*.

setOutputTraced:

– (void)**setOutputTraced:**(BOOL)*flag*

Causes the data (PostScript code, return values, and so on) flowing between the application’s single context and the Display PostScript server to be copied to diagnostic output.

setSynchronized:

– (void)**setSynchronized:**(BOOL)*flag*

Sets whether the **wait** method is invoked each time the receiver sends a batch of output to its destination.

setTextProc:

– (void)**setTextProc:**(DPSTextProc)*proc*

Sets the context’s text callback function to *proc*.

setWaitCursorEnabled:

– (void)**setWaitCursorEnabled:**(BOOL)*flag*

Sets whether the wait cursor is enabled or disabled according to *flag*.

See also: PSsetwaitcursorenabled()

startWaitCursorTimer

– (void)**startWaitCursorTimer**

Generates a pseudo-event to start wait cursor timer.

See also: setWaitCursorEnabled:

textProc

– (DPSTextProc)**textProc**

Returns the context’s text callback function.

unchainContext

– (void)**unchainContext**

Unlinks the child context (and all of it’s children) from the receiver’s list of chained contexts.

updateNameMap

– (void)**updateNameMap**

Updates the context's name map from the client library's name map.

wait

– (void)**wait**

Waits until the NSDPSContext's destination is ready to receive more input.

writeBOSArray:count:ofType:

– (void)**writeBOSArray:**(const void *)*data* **count:**(unsigned int)*items* **ofType:**(DPSDefinedType)*type*

Write an array to the context's destination as part of a binary object sequence. The array is taken from *data* and consists of *items* items of type *type*.

writeBOSNumString:length:ofType:scale:

– (void)**writeBOSNumString:**(const void *)*data* **length:**(unsigned int)*count*
ofType:(DPSDefinedType)*type* **scale:**(int)*scale*

Write a number string to the context's destination as part of a binary object sequence. The string is taken from *data*as described by *count*, *type,* and *scale*.

writeBOSString:length:

– (void)**writeBOSString:**(const void *)*data* **length:**(unsigned int)*bytes*

Write a string to the context's destination as part of a binary object sequence. The string is taken from *bytes* (a count) of *data.*

writeBinaryObjectSequence:length:

– (void)**writeBinaryObjectSequence:**(const void *)*data* **length:**(unsigned int)*bytes*

Write a binary object sequence to the context's destination. The sequence consists of *bytes* (a count) of *data*.

writeData:

– (void)**writeData:**(NSData *)*buf*

Sends the PostScript data in *buf* to the context’s destination.

writePostScriptWithLanguageEncodingConversion:

– (void)**writePostScriptWithLanguageEncodingConversion:**(NSData *)*buf*

Writes the PostScript data in *buf* to the context’s destination. The data, formatted as plain text, encoded tokens, or a binary object sequence, is converted as necessary depending on the language encoding of the receiving context.