UNIX Open API

UOAPI Overview

Changes made to WordPerfect for UNIX for the 7 version made it necessary to change the application programming interface as well. For example, both the character-based and graphical versions of WP 7 now have fully tokenized commands. The interface is now called the UNIX Open Application Programming Interface, or UOAPI. The WP 5.1 interface was known as the third-party interface.

The design of the UOAPI lets software developers write programs that interface directly with WP 7. (This same application interface will eventually be available for all 7-level WordPerfect Corporation products for UNIX.)

The two main parts of the interface specification are detection and communication. First, WP 7 must be able to detect that another program wants to interface with it. Once communication is established, the other program monitors, and perhaps modifies, the user input to WP 7. All user-generated commands are first converted to tokens (numeric command representations) and then passed to the interface program. The interface program can return the token unaltered, tell WP 7 to delete the token, or replace the token with a different token.

In addition to all the token commands, there are special tokens for getting status information and setting variables in WP 7. The polling token is particularly important. Normally an interface program is initiated by some user-generated token. However, there are times when an application wishes to automate WP 7, where no user interaction would take place. This is what the polling token is used for. When WP 7 is inactive, it will pass a special token to the interface program so that the interface program can perform a task while there is no user input.

Some tokens require added information. For example, the retrieve file token needs the name of the file to be retrieved. This extra information is contained in the token's parameters. Some tokens have no parameters while others have many.

All of the codes covered in this documentation are documented in bytes. For example, a short integer value (16-bit value) is defined as two bytes because of the different versions of UNIX available. All 16-bit and larger tokens/parameters are in Intel order. In this order the low byte of the value

comes first, then the high byte comes second. Some versions of UNIX follow this order and some versions follow the reverse order. Always use the Intel format when passing data larger than one byte.

Programming Concerns

Since character-based WP 7 is not designed to work in the background, redirecting standard input and output may cause problems. The table below describes the conditions that exist while character-based WP 7 is running.

Condition	Description
standard in	WP turns off the mapping of carriage return to line feed and disables the line discipline (meaning that it doesn't wait for a new line to complete the read).
standard out	WP disables all character conversions.

The minimum number of characters read is set to 1 and the default delay between characters is set to zero-hundredths of a second.

On machines that have pseudo TTYs, WP 7 turns off parity generation and forces an 8-bit character.

GUI WP 7 does not do text I/O except for a few startup error messages. Redirecting standard input and output will not accomplish anything, and may also cause problems. GUI WP 7 can be run as an icon with the standard -iconic command line switch.

Word Strings

A WordPerfect word string consists of 2-byte characters using the WordPerfect character sets. Each character is stored in Intel order (low-order byte first). The high byte identifies the WP character set the character belongs to. The low byte is the number of the character in that character set. The string must be terminated by a 2-byte null.

The WordPerfect zero character set contains all the standard ASCII characters. To reference standard ASCII characters, set the high byte to zero and the low byte to the standard ASCII code. You can view the WordPerfect character sets by selecting Character from the Insert menu in WP 7.

Detection

There are two startup methods for UOAPI programs. The UOAPI program can be specified in an initialization file called wpthird.ini and started by WP 7, or the UOAPI program can start WP 7 using a special command line switch. If a UOAPI program uses the command line method to start WP 7, WP 7 will not execute the programs from the wpthird.ini file. Both methods are explained below. (Both of these methods are the same as they were for the WP 5.1 interface.)

Initialization File Method

If WP 7 is started without the -third option, it reads the wpthird.ini file to determine if any UOAPI programs need to be started. The wpthird.ini file is found in the shlib10 directory under the WordPerfect installation directory. The wpthird.ini file is a text file that contains a line for each UOAPI program, formatted as shown below:

logical_name:path/filename

logical_name is some arbitrary identifier of the program that could indicate its name or function. *path* is the path of the executable file. *filename* is the filename of the executable file. An example is:

grammar:/usr/local/bin/gcheck

In this example the *logical_name* is "grammar." The *path* is "/user/local/bin" and the *filename* is "gcheck". The *logical_name* allows you to have two or more entries in the wpthird.ini file that reference the same executable. This is done by using a different *logical_name* in each reference. This allows an executable to perform different functions based upon the name by which it is called.

WP 7 executes each UOAPI program in turn, using the path from the wpthird.ini file. WP 7 passes two command line parameters to the UOAPI program. The parameters are the file descriptors of the pipes to use for communication with WP 7 and are accessible to the UOAPI program as argv[1] and argv[2]. The UOAPI program "reads" the descriptor argv[1]

for communications from WP 7 and "writes" to the descriptor argv[2] to send communications to WP 7. The descriptors are ASCII strings and must be converted to integers for use in read( ) and write( ) system calls.

The only way multiple UOAPI programs can be detected by WP 7 is when they are called through the wpthird.ini file. The maximum number of UOAPI programs that can be put in the wpthird.ini file is eight.

Command Line Method

Before starting WP 7, the UOAPI program needs to use the pipe( ) system call to create two pipes for passing messages back and forth with WP 7.

The UOAPI program then uses the fork( ) and exec( ) system calls to start WP 7. The pipe handles are given to WP 7 as command line arguments. To accomplish this, add the following to the command line when your program starts WP 7:

-third *xxxxx,yyyyy*

The table below describes the arguments shown in the command line.

Argument	Description
xxxxx	The ASCII representation of pipe handle value that WP 7 uses to read messages from the UOAPI program.
yyyyy	The ASCII representation of pipe handle value that WP 7 uses to send messages to the UOAPI program.

Communication

Like its WP 5.1 counterpart, the WP 7 interface works on a cause and affect relationship between itself and your UOAPI program. Your UOAPI program cannot send a communication to WP 7 unless it has first received some communication from WP 7.

Another important fact about interface communication is that the information packets sent through the pipes do not come in one complete chunk. They come piece by piece. In each case, the first field of the packet data indicates the total length of the remaining packet data. This field only indicates the length of the remaining packet data and not its own length. Make sure you always use the block read/write functions supplied in the sample UOAPI program (see *Program Files* later in this section), or similar functions, to get and send pipe information. These functions wait until all of the information in a given communication has been delivered before handing the communication to your interface program. (The WP 5.1 interface worked in this same way.)

Initializing

The first communication sent to the UOAPI program is an initialization packet. The figure and table below describe the packet format.

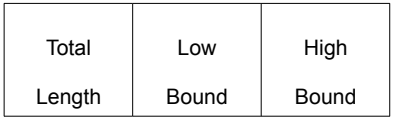
Total Length	Token	Name Length	Name	Product	Major	Minor	Language Code 1	Language Code 2

Name	Size	Description
Total Length	2 bytes	Two bytes stored in Intel order that indicate the total length of the remaining packet data.
Token	2 bytes	Two bytes stored in Intel order that contain THIRD_INIT_TOK. This token value identifies the initialization packet.
Name Length	2 bytes	Two bytes stored in Intel order that indicate the length of the name field.
Name	Variable	Null-terminated byte string that contains the logical name of the UOAPI program as read from the wpthird.ini file.
Product	1 byte	One byte identifying the WordPerfect application that started the UOAPI program (3 = GUI WP 7, 4 = character-based 7).
Major Version	1 byte	One byte identifying the major version of the WordPerfect application. (The major version number of WP 7 = 6.)
Minor Version	1 byte	One byte identifying the minor version of the WordPerfect application (The minor version number of WP 7 = 0.)
Language Code 1	1 byte	First byte of the 2-byte language string identifying the language of the current invocation of WP 7 (“u” for US English).
Language Code 2	1 byte	Second byte of the 2-byte language string identifying the language of the current invocation of WP 7 (“s” for US English).

If your UOAPI program calls WP 7 using the command line method, the string value in the Name field of the initialization packet will be "exec".

After receiving the initialization packet, the UOAPI program acknowledges it can communicate by returning a data packet. The figure

and table below describe the format of the data packet.



Name	Size	Description
Total Length	2 bytes	Two bytes stored in Intel order indicating the total length of the data following (always 4).
Low Bound	2 bytes	Two bytes stored in Intel order indicating the lower bound of the token range the UOAPI program wants to see.
High Bound	2 bytes	Two bytes stored in Intel order indicating the upper bound of the token range the UOAPI program wants to see.

If the UOAPI program does not want to communicate with the WordPerfect application, it sets the length to 4 and both the low and the high bound values to zero. For example, if your program wants to communicate only with the character-based version of WP 7, you should use this method to avoid the GUI version.

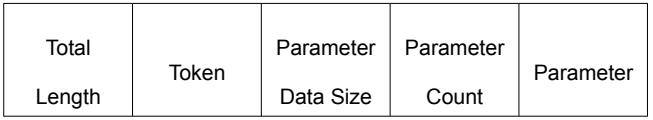
If your UOAPI program needs to see all tokens, it should set the total length to 4, the low bound to 0 and the upper bound to 0xFFFF.

Setting the token bounds appropriately can affect performance significantly. It helps WP 7 to avoid incurring the Inter-process Communication (IPC) overhead associated with sending unnecessary tokens to the UOAPI program.

Token Handling

After every keystroke, menu selection, or mouse event, the UOAPI program receives a token data packet from WP 7. Each token data packet consists of a Total Length field followed by a token and the token's

parameters, if any. The figure and table below describe the format of a token data packet.



Name	Size	Description
Total Length	2 bytes	Two bytes stored in Intel order indicating the total length of the remaining packet data.
Token	2 bytes	Two bytes stored in Intel order containing a WP 7 token.
Parameter Data Size	2 bytes	Number of bytes of parameter data (includes the Parameter Count field and the Parameter field).
Parameter Count	2 bytes	Number of parameters in the Parameter field (field doesn't exist if parameter data size is 0).
Parameters	Variable	Token's parameters (field doesn't exist if parameter data size is 0).

The UOAPI program examines each token data packet and returns a token data packet to WP 7 using the same format. The UOAPI program can pass the data back unchanged, replace the token with another token, or delete the token. UOAPI programs can replace tokens by simply returning the token data packet of the new token. To delete a token, the UOAPI program returns a token data packet with the Total Length field set to zero.

If WP 7 is communicating with multiple UOAPI programs, it does so according to a specific sequence. The order in which the UOAPI programs appear in the wpthird.ini file determines the sequence of communication. When a token is generated in WP 7, it sends a token data packet to the first UOAPI program in the sequence. When WP 7 receives a token data packet from the first UOAPI program it sends it to the second UOAPI

program, and so forth until the sequence is completed. WP 7 will execute the token returned by the last UOAPI program. If any UOAPI program deletes a token, the sequence is ended with that UOAPI program. WP 7 will not send a token data packet where the Total Length field is zero. WP 7 never sends or receives multiple token data packets and each packet contains only one token. If your UOAPI program needs to send multiple commands to WP 7 it should send a token instructing WP 7 to run a file macro.

For example, three UOAPI programs have been loaded by WP 7 from the wpthird.ini file. WP 7 passes a token (token data packet) to UOAPI:1. UOAPI:1 processes the token, and passes it back to WP 7. WP 7 passes this token to UOAPI:2. UOAPI:2 deletes the token. When UOAPI:2 passes its empty token data packet to WP 7, WP 7 ends the sequence at that point. UOAPI:3 will never have the empty token data packet passed to it.

This type of command (token) handling is similar to that of the WP 5.1 interface. However there are two major differences. The WP 5.1 interface passed tokens for GUI WP and key codes for character-based WP. WP 7 passes all commands as tokens in both the character-based and GUI versions. The second difference is the number of tokens that can be passed. WP 5.1 could pass multiple tokens in one communication. All WP 7 token data packets have only one token.

Polling

When WP 7 is idle it will send a polling token to the UOAPI program. This gives the UOAPI program the ability to control WP 7 without normal user input. The polling token is formatted as a normal token data packet with the value THIRD_POLLING_TOK in the Token field and zero in the Parameter Data Size field. The UOAPI program can return the polling token unchanged or replace it with another token. WP 5.1 did not have polling.

Manipulating Variables

WP 7 has two types of variables, system variables and merge variables. System variables are set by WP 7 and can be read using macros, merges and UOAPI programs. They provide information about the current status of WP 7. Merge variables can be created, set, and read by macros, merges

and UOAPI programs. Merge variables can be used to pass information between macros and UOAPI programs. The UserFunction macro command can also be used to pass information from a macro to a UOAPI program. See WordPerfect's online Macros Help for more information the UserFunction macro command. (online Macros Help can be accessed by selecting Macro from the Help menu.)

Reading System Variables

WP 5.1 passed each communication to your interface with 32 state bits as the second field of the communication. These state bits had basic information about the status of WP 5.1. For example, a macro is running, the main editing screen is active, or one of the 9 main editing screens is active. You could also send the {SYSTEM} macro command, followed by information telling WP what system information you wanted, to WP through the chain of interface programs. When WP executed this macro command, it would return the requested information back through the chain of interface programs to your interface program.

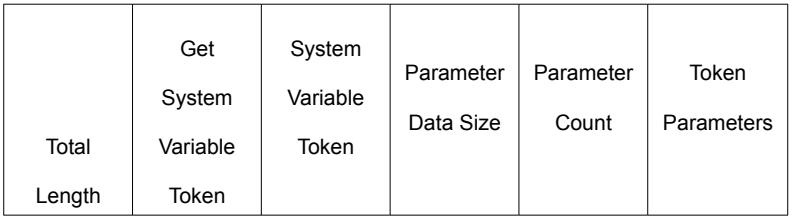
In WP 7, UOAPI programs can get all status information about WP 7 through system variables. For example, your program can test if printing is in progress, if WP 7 is in an edit structure (headers and footers), if Block is active, what the current text attribute is, what the current page number is, and so on. All system variable read requests are passed directly to WP 7 and all responses are passed directly back to the requesting UOAPI program.

To access system variable data, UOAPI programs send a data packet to WP 7 formatted as shown in the figure and table below.

Total Length	Get System Variable Token	System Variable Token

Name	Size	Description
Total Length	2 bytes	Two bytes stored in Intel order indicating the total length of the remaining packet data.
Get System Variable Token	2 bytes	Set to THIRD_GET_SYSTEM_TOK.
System Variable Token	2 bytes	Token value of the desired system variable.

In response to the Get System Variable request, UOAPI programs receive a data packet formatted as shown in the figure and table below.



Name	Size	Description
Total Length	2 bytes	Two bytes stored in Intel order indicating the total length of the remaining packet data.
Get System Variable Token	2 bytes	Set to THIRD_GET_SYSTEM_TOK.
System Variable Token	2 bytes	Token value of the system variable being returned.
Parameter Data Size	2 bytes	Number of bytes of parameter data (includes the parameter count and parameter fields.)
Parameter Count	2 bytes	Number of parameters in the parameters field.
Parameters	Variable	The token parameter values.

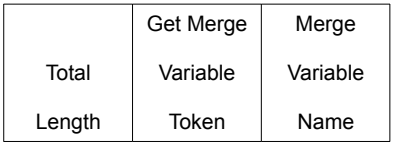
If the variable is not accessible, the *Parameter Data Size* field will be zero.

Reading Merge Variables

Variables can be created in WP 7 and assigned an initial value. They can be set and read by merges, macros, and UOAPI programs. They were called merge variables in WP 5.1. In WP 7, they are called merge variables or persistent variables.

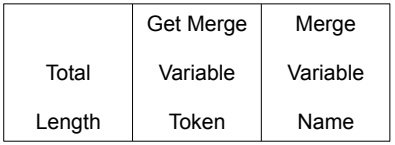
WP 7 merge variables are not purged from memory when macros or a merge has ended. They exist from the time they are created until WP 7 terminates. This enables communication between UOAPI programs, merges and macros. For example, a UOAPI program can read and set merge variables from the open application interface, and have them persist through macros and merges.

To get the value of a merge variable, a UOAPI program sends a data packet to WP 7. The figure and table below describe the packet format.



Name	Size	Description
Total Length	2 bytes	Two bytes stored in Intel order indicating the total length of the remaining packet data.
Get Merge Variable Token	2 bytes	Set to THIRD_GET_MERGE_TOK.
Merge Variable Name	Variable	Null-terminated word string containing the name of the desired merge variable.

In response to the Get Merge Variable request, the UOAPI program will receive a data packet. The figure and table below describe the data packet format.

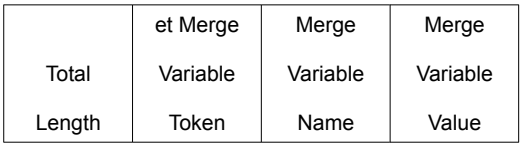


Name	Size	Description
Total Length	2 bytes	Two bytes stored in Intel order indicating the total length of the remaining packet data.
Get Merge Variable Token	2 bytes	Set to THIRD_GET_MERGE_TOK.
Merge Variable Value	Variable	Null-terminated word string containing the value of the desired merge variable.

If the requested variable does not exist, a null value will be returned in the *Merge Variable Value* field.

Setting Merge Variables

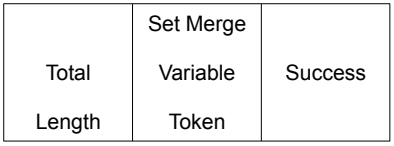
UOAPI programs can set merge variables from the open application interface by sending a data packet to WP 7. The figure and table below describe the packet format.



Name	Size	Description
Total Length	2 bytes	Two bytes stored in Intel order indicating the

Name	Size	Description
		total length of the remaining packet data.
Get Merge Variable Token	2 bytes	Set to THIRD_SET_MERGE_TOK.
Merge Variable Name	Variable	Null-terminated word string containing the name of the desired merge variable.
Merge Variable Value	Variable	Null-terminated word string containing the value of the desired merge variable.

In response to the Set Merge Variable request, the UOAPI program will receive a data packet. The figure and table below describe the packet format.



Name	Size	Description
Total Length	2 bytes	Two bytes stored in Intel order that indicating the total length of the remaining packet data.
Set Merge Variable Token	2 bytes	Set to THIRD_SET_MERGE_TOK.
Success	1 byte	One byte indicating the success of the set operation. Zero for success or one for failure.

Terminating

When the user exits, WP 7 sends a token data packet containing THIRD_EXIT_TOK in the token field. The UOAPI program should use this as the signal to terminate.

IMPORTANT: Character-based WP 7 saves standard in, standard out, and other system information as it was at the time WP 7 was executed. When WP 7 terminates, it does not reset this information, but restores it to the settings it saved at the start of execution. This is particularly important for UOAPI programs using the command line method to call WordPerfect

Program Files

The WordPerfect for UNIX Software Developer's Kit diskette contains a sample program that uses the interface described above to communicate with WP 7. The uoapi.c program is a sample UOAPI program written in the C programming language.

All of the WP 7-specific data and all of the legal tokens have been defined in the uoapi.h file. This file can also be found on the UNIX SDK diskette. Use this file to put tokens into your UOAPI programs.

The UWPMRS.WP file is a WP 7 document file that documents each of the legal WP 7 tokens, their purpose, their parameters, and parameter format. This file can also be found on the

UNIX SDK diskette.

Token parameters generally follow the format described in the table below.

Size	Description
1 byte	Type of data to follow as defined in M_TYPE_MASK. See below.
Variable	Parameter data.

This format would be repeated for each parameter of a given token. The actual data and type for each parameter of a token is documented in UWPMRS.WP. The table below describes the M_TYPE_MASK field of the parameter information.

M_TYPE_MASK	Size of Following Data
M_WSBYTE	1 byte (signed).
M_WUBYTE	1 byte.
M_WCHAR	1 byte (character).
M_WSWORD	2 bytes (signed short integer).
M_WUWORD	2 bytes (unsigned short integer).
M_WSDWORD	4 bytes (signed long integer).
M_WUDWORD	4 bytes (unsigned long integer).
M_SWPU	4 bytes (signed).
M_WPU	4 bytes.
M_WPCSTRING	Variable (Null-terminated word string format).
M_STRING	Variable (Null-terminated ASCII string format).
M_FLOAT	8 bytes (real number, the data type is double in host sp

Like all previous data, information larger than 1 byte is stored in Intel order. M_SWPU and M_WPU types mean the value of the data to

follow is in WordPerfect units. A WordPerfect unit is defined as one 1200th of an inch.