Writing Tool Session Overview

A *writing tool session* is initiated with a writing tool by the client application. In most cases, this occurs when the user selects the tool from a menu within the client. After the writing tool has been loaded into memory it becomes the foreground application. It presents the user interface and sends the appropriate queries and commands to the client application. The client replies with the queried information or fulfills the writing tool commands. The API defines a writing tool session between a single client document or window and a single writing tool window. Multiple sessions between clients and writing tools must be managed by the individual applications.

The following sections describe the steps necessary to execute a writing tool session. Installation of the writing tool into the client must take place before the session can begin. This gives the client access to information that enables it to initiate execution of the writing tool. The API defines a group of messages that are used to process the text of the client. There are message groups which handle the initialization between the tool and the client, determine the parameters of operation for the session, and handle the transfer of writing tool specific data. Additional groups handle the transmission of text from the client to the writing tool, text correction, and editing during the session. A final group handles the termination of the writing tool session.

Installation and Initiation

The Writing Tools API defines the minimum set of platform-independent requirements for implementation. These are: writing tool installation, user selection, writing tool location, writing tool communication, and writing tool service identification. UNIX-specific information necessary for installation is contained in *UNIX Platform-Specific Information* later in this section.

The installation process consists of appending information to a global file or database that is accessible to all writing tools and clients. Each tool

writes one record to this file for each logically separate writing tool service. However, it should not add entries for functions within a single writing tool. For example, if a writing tool acts as both a speller and a thesaurus, these two functions may appear as separate menu items. The client will use the corresponding information for each function. However, if the speller contains options to verify the current word or the entire document, separate menu items for word and document checking should not appear within the client.

User selection means that the client has the information necessary to provide access to the writing tool through its user interface. This could include menu strings, long descriptions, button names, and so forth.

The client must also be able to locate the writing tool. This information may take the form of a network address or a disk path to the executable.

The writing tool communication requirement allows clients and writing tools to converse through inter-process communication. Examples of this information are process names and addresses.

The writing tool service information describes the writing tool service type to both the client and the writing tool. If a writing tool supports multiple functions, it must be able to determine which function is desired by the user. The client may also wish to search the file for a specific writing tool function. This information should also be available.

Sending Writing Tools API Messages

The Writing Tools API is implemented as a layered communication protocol. The highest level of this protocol is the C-callable interface. C functions are defined for sending and receiving API messages.

The first set of functions is used for sending API command messages, which are messages that don't require a response from the other application. The Writing Tool Command messages all have a WTC_ prefix. Writing tools use the wtcTlSend( ) function, while clients call the wtcClSend( ) function.

The next set of functions are used for query messages. The Writing Tool Query messages expect a corresponding reply from the other application and begin with a WTQ_ prefix. Writing tools use the wtqTlSend( ) function, and clients use the wtqClSend( ) function. These functions also contain parameters to store the Writing Tool Response messages, which all have a WTR_ prefix.

Applications define a callback function to receive Writing Tools API messages. Writing tools define the wtTlReceive( ) function, while clients define the wtClReceive( ) function. These functions receive both command and query messages. Replies to query messages are stored in message buffers provided to the application as part of the function call.

Platforms may define additional C functions as needed to support communication or other aspects of the Writing Tools API.

Initiating a Writing Tool Session

The writing tool session is initiated when the client sends the WTC_INIT message to the writing tool. This may occur in response to a WTC_RQINIT message from the writing tool, or by the user selecting the writing tool from a client menu.

The WTC_INIT Message

The client application initiates a writing tool session by sending the writing tool a WTC_INIT message. This message indicates the desired mode of execution for the writing tool, as well as the version of the API under which the client operates.

The WTC_RQINIT Message

Some writing tools or operating platforms may support or require a writing tool to be able to initiate the writing tool session. The writing tool does this by sending the client a WTC_RQINIT message. This message is very dependent on operating system functionality, but should contain the information the client needs to initiate the writing tool session.

Customizing API Communication

After the writing tool has received the WTC_INIT message, it may send messages that customize the parameters of communication. These include the WTQ_INFOBLOCK message and the WTQ_UNITINFO message. These messages are normally sent only once at the beginning of each writing tool session. However, during client activation, the user may modify some of the information that has been sent to the writing tool. The writing tool will receive notification when this occurs, and may then re-issue these messages to request the updated information. If the writing tool wishes to operate by the default parameters of the API it does not need to send these messages.

The WTQ_INFOBLOCK Message

The writing tool sends a general setup information block in the WTQ_INFOBLOCK query message. The general setup information includes the mode of text representation, supported tool data areas, and

user interface languages available, as well as other initialization parameters.

The WTR_INFOBLOCK Message

The client replies to the query with a WTR_INFOBLOCK message with information indicating what options are available for this writing tool session.

The WTQ_UNITINFO Message

The WTQ_UNITINFO message requests information on the text units available in the client. This is for the purpose of presenting the user with the appropriate menu choices. If a writing tool does not need this information, this message does not need to be sent.

The WTR_UNITINFO Message

The client replies with the WTR_UNITINFO message. The parameters to this message indicate which units are supported by the client, which units are currently available, and which unit the client considers to be the default unit.

Tool Data Areas

The writing tool data areas allow writing tools to store data specific to their product with the client application. Data may be stored in a general client product area, in the client's current document area, or at the client's cursor position. The client specifies which of these storage modes it supports in the *tdamode* field of the WTR_INFOBLOCK reply message. The writing tool requests a tool data area by sending a WTQ_READTOOLDATA query message. The client returns the data using a WTR_READTOOLDATA reply message. The writing tool uses the WTQ_WRITETOOLDATA command to transfer a tool data area to the client. The client replies with the WTR_WRITETOOLDATA message.

Tool Data Storage Modes

The Writing Tools API defines three storage modes for writing tool data.

The first of these modes specifies that the data is stored in a product area independent of the current document. This data can then be retrieved during any writing tool session with that client. This area is useful for client-specific setup information.

The second mode specifies that the data is stored in a document data area. For instance, WordPerfect will store a document-specific supplemental dictionary in this area for the WordPerfect Speller.

The third mode specifies that the tool data is stored at the current cursor location. A reference to the tool data area will be sent as a document object in the text and codes buffer, but only if the current text mode allows for formatting and object information. This area allows writing tools to store notes or other information at any location in a document. Tool data for this mode may create a visual marker in the client's document. It may also be specified as temporary to the writing tool session, or as a permanent part of the document. This type of tool data may also be used to mark sections of text as belonging to the writing tool.

Text Transmission

The writing tool initializes a text query with the client with the WTQ_TEXTBLOCK message. The client returns the text with the WTR_TEXTBLOCK message. The writing tool may use the WTQ_NEXTTEXTBLOCK message to request subsequent text blocks in the same query.

The WTQ_TEXTBLOCK message contains parameters that specify the text the writing tool wishes to process. This specification is given in terms of the units designated by the client in the WTQ_UNITINFO query.

The WTR_TEXTBLOCK message contains information regarding the text

it is transmitting to the client. The text is also returned. If all requested text is returned in the reply message the *complete* flag is set. Otherwise, the writing tool sends the WTQ_NEXTTEXTBLOCK messages to request the next block of text. This continues until all blocks have been sent.

Certain file formats may support a form of *text objects*. Text objects are references to larger portions of the text that are not contained in the normal text stream. For example, a font change code might be contained in the normal text stream, but the details of the new font might not. Writing tools can use the WTQ_READTEXTOBJECT message to request this information. The client replies with the WTR_READTEXTOBJECT reply. Similarly, the WTQ_WRITETEXTOBJECT message allows the tool to replace or append to the object. The client replies with the WTR_WRITETEXTOBJECT message.

Text Correction

The API defines text locations in terms of an API cursor position. This cursor is not related to the input cursor location of the client. After each text block is returned, the API cursor is at the end of that text block.

At any time during a query, the writing tool may wish to make additions or corrections to the text. The tool must first specify the position at which the change will take place. It does this using the WTQ_GOTO message.

The WTQ_HILITE and WTQ_DEHILITE messages instruct the client application to add or remove highlighting from the client's document window. Highlighting takes place at the API cursor location.

The WTQ_REPLACE message is used to replace text in the client buffer. The message specifies how many bytes (in terms of the current text format) should be deleted from the client's edit buffer. It also transmits a buffer of text to be added to the client's buffer. The client responds with the WTR_REPLACE message. The writing tool may use the WTQ_UNDOREPLACE message to undo the last replacement.

User Editing

In the text correction stage discussed above, the client's document is corrected through API messages. This method is suitable for localized corrections. There are instances, however, when the user will wish to make larger-scale revisions to the text without completely terminating the writing tool session. The Writing Tools API allows the user to make editing changes in the client document and then return to the writing tool session with only minimal interruption in the information flow.

The user initiates editing within the client by making it the foreground application. The writing tool may also initiate this action with the WTC_RQCLACTIVE message. When the client receives control, it sends the WTC_CLACTIVE message to inform the writing tool. The user makes the changes, then brings the writing tool to the foreground again. The writing tool sends a WTQ_TLACTIVE message to the client. The client replies with the WTR_TLACTIVE message, informing the writing tool of changes made during the editing session. If supported by the platform, applications also send messages upon losing activation. Writing tools send the WTC_TLINACTIVE message, while clients send the WTC_CLINACTIVE message.

The parameters in the WTC_CLACTIVE message inform the writing tool of the action it should take. This may be to keep its window visible during the editing session, or to hide in the background. Clients may send this message multiple times to change among the various states of the writing tool. The WTC_CLACTIVE message may also inform the writing tool that it should regain activation.

When the writing tool regains control, it sends a WTQ_TLACTIVE message to the client. The client responds with a WTR_TLACTIVE message. The parameters to this message specify which blocks have been changed by the user editing, and if anything has changed that would require new information blocks to be passed to the writing tool.

Termination

Termination of the writing tool session is always completed by the writing tool. The tool accomplishes this by sending the WTC_TLTERM

command message. After this message is sent, no other writing tool communication can take place.

When the client wishes to terminate the session, it sends the WTC_RQTLTERM message to the writing tool. The writing tool may then send any number of messages, such as the WTQ_WRITETOOLDATA message, before terminating the conversation with a WTC_TLTERM message.