W3A – WWW Applets API, common functions

May 30, 1995

W3A (common part): Functions exported by the browser

The browser starts applets and feeds them data, but it will also execute tasks on behalf of the applets, such as retrieving a new document and displaying a previous document.

The file `w3a.h` defines the structs `W3ADocumentInfo` and `W3ABrowserInfo`:

```
typedef struct {
    char *url;
    char *mime_type;
    char *mime_params;
    char *title;
    char *referer;      /* NULL if not known */
    char *status;       /* e.g. "403 not found" */
    char *location;     /* For redirect if status = 3xx */
    long size;          /* in bytes, -1 if unknown */
} W3ADocumentInfo;
typedef struct {
    char *version;        /* browser name & version */
    int nformats;         /* # of accepted formats */
    char **formats;       /* accepted formats */
    float *preferences;   /* Each in (0,1] */
} W3ABrowserInfo;
```

When a function changes a field in `W3ADocumentInfo`, it must free the previous contents and allocate new space on the heap.

The type `W3AWindow` is defined in the `w3a.h` file to be the correct type for the platform under which the browser and applets run. Under X it will be an alias for `Widget`, on other systems it will be something else.

In the following, reference is made to a type `Bool` and constants `FALSE` and `TRUE`. In the `w3a.h` file, they are defined as macros for `int`, 0 and 1, respectively.

All browsers that support at least one of W3A/A, W3A/F, W3A/V, W3A/P, and W3A/U, must also provide the following functions:

- `Bool W3Aprocess(W3ADocumentInfo *info, int method, const char *data, size_t nbytes)`

This is the function that an applet calls when it wants to follow a hyperlink, for example, when the user clicks in a viewer. When the *method* argument is one of GET_METHOD or DELETE_METHOD, the data and nbytes arguments are not used. The other two methods, PUT_METHOD and POST_METHOD, require that data holds the data to be sent to server in the body of the request.

In the *info* argument the url, title and referer fields must be initialized, but the latter two may be NULL. All other fields must be set to NULL (and size to -1). On successful return all other fields will have been filled in (and title and size may have changed). When the function returns with FALSE, errno will have been set and status may have have been filled in with a (human readable) status response, such as returned, e.g., by HTTP servers.

There are two possible semantics for this function In a browser without parallel threads, the function returns when the document has been fully received and is being displayed or when the retrieval is broken off (by the user, or because of an error). The returned value indicates the success of this operation.

In a browser that is multi-threaded, the browser may retrieve just enough to be able to start the right viewer, and then do a fork(), leaving the task of copying incoming data to a child process. The returned value only indicates that the initial part of the document was received OK.

> **Note**: *The latter option, while allowing a faster turn-around to the user, may have a few complications: we cannot expect all viewers to be written for concurrent operation, therefore the browser must buffer the data and only call the viewer's* writeXXX() *function from the main thread, because otherwise it could happen that the viewer is analysing its data while new data is being added at the same time. Also, how expensive is it to do a* fork() *in a program as large as the browser?*

Important: a browser may wish to close the previous viewer before opening another one, which means that a viewer instance that calls W3Aprocess may not 'exist' anymore when the call returns. More precisely: the viewer may have received a closeXXX call in the mean time, destroying windows and datastructures.

> **Question:** *Maybe* W3Aprocess *needs another argument: the ID of the viewer that calls it. That way the browser knows in which window the hyper-jump originated, which could be useful for a browser with multiple top-level windows.*

- long W3Asubprocess(W3ADocumentInfo *info, int method, const char *data, size_t nbytes, W3AWindow window)

  This function may be called by a viewer when it want to retrieve and view a sub-document, such as an in-line image. The browser will retrieve the document (by itself or via the appropriate agent) and will display it (again by itself or via the appropriate viewer), but it will not open a new window for it. Instead, it will pass the window on to the viewer.

  The return code is the id of the viewer that is displaying the document, or -1 in case of an **error**.

The browser will *not* notify applets via the event broadcasting mechanism (`W3Aevent()` below) that a new document is opened. Also, when the browser closes viewers, it will not close the viewer for this subdocument. It is the task of the viewer that called `W3Asubprocess` to close the sub-viewer (by calling `W3AcloseView`, see below.)

- `int W3AopenDoc(const char *url, int method, int flags, const char *referer)`

  Directs the browser to retrieve (method = GET_METHOD), put (PUT_METHOD), or post (POST_METHOD) a document, without displaying it. The function returns a file descriptor that must be used in calls to `W3AreadDoc()` or `W3AwriteDoc()` and it may also be used in calls to `select()`

  The *`flags`*, if not 0, modify the behaviour of this call and the `W3Aread` and `W3Awrite` functions. On Unix, the only possible flag is currently `O_NONBLOCK`, which causes the functions to return immediately, without waiting for the operation to complete.

  `referer` is the URL of the document that contains *`url`* and from which the function was called. It may be `NULL` if no such URL is available.

> **Question:** *The caller of this function (probably a viewer wanting in-line images) may wish to specify the particular formats that it prefers. The HTTP protocol allows a single URL to stand for a family of documents each in a different format. How should the caller communicate its desires to the browser? Via an extra argument or a variable-length list of arguments:* `format1, pref1, format2, pref2,...`*?*

- `Bool W3AinfoDoc(int fd, W3ADocumentInfo *info_return)`

  Provide information about a document being retrieved. When a failure occured while opening, reading or writing the document, this function can also be called to get the reason: it will be recorded in the `status` field.

- `int W3AreadDoc(int fd, char *buf, size_t bufsiz)`

  The *`fd`* argument must be a file descriptor returned by `W3Aopen()`. The return value is the number of characters read, or -1 in case of error. A return of 0 means end of file.

  When the `O_NONBLOCK` flag was set in the call to `W3Aopen`, the function returns immediately, whether or not characters were available. In the latter case, the return code is -1 and `errno` is set to `EAGAIN`.

- `int W3AwriteDoc(int fd, const char *buf, size_t nchars)`

  The function tries to write *`nchars`* and returns the number of characters actually written. On Unix, there are two cases when that number may be less than *`nchars`*: when the operation is interupted by a signal or when the `O_NONBLOCK` flag has been set.

  A return of -1 indicates an error, except when `errno = EAGAIN`; the latter can only happen when `O_NONBLOCK` has been set and it indicates that the call could not be satisfied immediately, but that the caller should try again.

- `Bool W3AcloseDoc(int fd)`

  Closes the connection and invalidates the file descriptor.

- `long W3AopenView(const W3ADocumentInfo doc, W3AWindow window)`

  A viewer can call this function to open another viewer in a sub-window. The browser will start the appropriate viewer and return its ID, or -1 in case of an error.

- `int W3AwriteView(long id, const char *buf, size_t nchars)`

  This function is used to send data to be displayed to a viewer previously opened with `W3AopenView`. The return code is equal to `nchars` or -1 in case of an error.

- `Bool W3AcloseView(long id)`

  Closes a viewer previously opened with `W3AopenView`.

- `int W3AresolveURN(const char *urn, char ***url_list_return)`

  Currently returns -1, indicating that the URN could not be resolved.

- `void W3AbrowserInfo(W3ABrowserInfo *info)`

  Returns information about the browser and its capabilities. For example, an HTTP agent needs this to construct the 'Accept:' headers required by the HTTP protocol.

> **Question:** *The information may change, depending on context (see* **the note on** `W3Aopen`*). Maybe this function needs an extra* `fd` *argument by which the agent identifies itself?*

- `void W3Aevent(long id, long eventtype, void *params)`

  When a viewer or user function has an event to report, it calls `W3Aevent`, which in turn calls the event functions of all other viewers and user functions. See the explanation below.

- `W3AWindow W3Atoplevel(void)`

  The `W3Atoplevel` function returns a window (widget) that can be used as a parent for pop-up windows. It should not be used as a parent for normal windows, since it might not be 'mapped' (i.e., visible on the screen).

Question: *There are many useful, though not essential functions that many applets share. The browser could offer them, or do we create* `libWWWutils` *and simply recommend that authors use it?*

Events and synchronization

In some viewers, events can happen that do not immediately cause a new document to be loaded, but that might cause other viewers to do something. For example, two viewers might be synchronized in such a way that a mouse click in one of them causes both windows to display a new image.

Since it not clear at this point what types of events must be supported, W3A only defines an abstract mechanism. The contents and the semantics of events will have to be standardized in some way, or there has to be a way for viewers to agree on events. This is a matter for a future revision.

The current mechanism is very simple: events are identified by a number and may have a single parameter in the form of an untyped pointer. There is no mechanism for filtering events. All events are treated the same and all viewers receive all events.

At the moment, there is only one defined event type, which is used by the browser itself. It has event code 1, and indicates that the browser (successfully) executed a `W3Aprocess`. Applets that are interested in keeping track of the 'current' document, such as hotlist or history applets, can use this. The event-dependent info will be a pointer to a `W3ADocumentInfo` structure, describing the document that was just opened for viewing. (Note that the browser may send this event more than once for the same document, for example when the document's title changes. Applets must be prepared for this.)

The browser functions as the central event dispatcher. It has a function `W3Aevent` that an applet calls when it has an event to report. The `W3Aevent` function then calls every viewer (except the originator of the event) with the same arguments that it received itself.

Every viewer and every user function must have an exported function `eventXXX`, with four arguments: the applet's ID (`long`), the ID of the applet in which the event originated (`long`), the event type (`long`), and a pointer to event-dependent info (`void *`). Simple viewers that don't need events must still supply an (empty) function.

Sub-viewers that have been opened by `W3AopenView` will also be called by the browser. This means that the viewer that opened the sub-viewers does not have to do so itself. But a parent viewer is free to call the `eventXXX` functions of its sub-viewers if it wants to.

Examples of situations where the event mechanism is usefule are the following:

- Clickable images, either in the old way (`ISMAP`) or the new (`FIG`) can be displayed with sub-viewers that do not know that the image they display is actually an anchor. Instead, they generate an event when the user clicks on them and call `W3Aevent` with the mouse coordinates as arguments. The parent viewer's `eventHTML` function will be called and it can process the event.

> **Question:** *it would be nice if the image viewer* did *handle the hyperlinks itself, though. Because it could then give some visual feedback, especially in the case of* `FIG` *images, where the hotspots are known. A special viewer is needed for this, but how are the hotspots communicated to that viewer?*

- Certain input fields in HTML forms could be handled by sub-viewers. A viewer for a scribble field, for example, could use the event mechanism to notify the HTML viewer that the user has started (or ended?) scribbling and pass a pointer to the resulting drawing as parameter.

- A slide show might want to use a clock to time the images and other displays. A special timer applet could be the source of the timing signals. The slide show viewer could open a sub-viewer of type 'application/x-clock' that generated a clock-tick event every few seconds.

Error codes

When the above functions fail, they set the *errno* variable. functions exported by applets should also set this variable when they fail. Apart form the values already defined by the OS libraries, the following four extra values are defined in `w3a.h`:

`EURL` Syntax error in URL

`EMETHOD` Illegal method

`ENYI` Not yet implemented

`ETYPE` No applet for this type

`EFORMAT` Error in data

`EUSER` Interrupted by user

The interpretation of `EUSER` depends on the context. It is recommended that an HTML viewer that encounters this error while retrieving in-line images should not retrieve any further images.

> **Note:** *Error handling is still a weak spot; something more has to be said about the proper use of* `errno`
