W3A/A – WWW Applets/Agents API

May 30, 1995

W3A/A: Agents

Agents are used to add new protocols to a browser. They are characterized by

1. one or more protocols (possibly given as a regular expression matching a URL, such as `"wais:*"` or `"new-proto:*"`).
2. the object file(s) to load (could be a list of object files or the name of a library).
3. a suffix, that is used to distinguish function names in this agent from functions in all other applets.

Agents are responsible for determining the (MIME) type of a document and also for decoding of compressed or otherwise encoded documents. In some case that may even entail using heuristic rules to guess the type (e.g., in case of the FTP protocol).

Agents are also responsible for authentication, encryption and decryption for protocols that require it.

> **Question:** *This is a difficult area, since it may involve user interaction, which is normally not a task for an agent. How can the user interface be decoupled from such an agent?*

All agents must define (and export) the following functions:

- `Bool initXXX(char ***protocols, int *nrprotocol)`

  (Where `XXX` is the abovementioned suffix.) Initializes private variables inside the applet and returns the list of protocols that the agent handles. When the function returns `FALSE`, the *errno* **variable** holds the reason.

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

  Opens a connection and returns a file descriptor that is valid in subsequent `readXXX`, `writeXXX` or `closeXXX` calls. The file descriptor may also be used in `select(2)` calls.

  The *method* is one of `GET_METHOD` (=0), `PUT_METHOD` (=1), or `POST_METHOD` (=2). The symbolic constants and types are defined in `w3a.h`.

The `flags` are platform dependent. On Unix, the only supported flag is `O_NONBLOCK`, which directs the agent to return as quickly as possible, without blocking forever until the request is sent. The flags also affects `readXXX` and `writeXXX` calls.

The `referer` should be set to the URL of the document that contained `url` as a hyperlink and from which this call was issued. It may be set to `NULL` if no such URL is available.

When the call fails, it returns `-1` and *errno* holds an **error code**.

- `int readXXX(int fd, char *buf, size_t nbytes)`

  Returns the number of characters copied to `buf`, or -1 in case of an error. A return value of 0 indicates end of data. The `self` argument must be one returned by a call to `openXXX` with method `GET_METHOD`.

  When the return code is -1, the variable *errno* holds the reason.

- `int peekXXX(int fd)`

  The `peekXXX` function returns the number of bytes that are available for reading with `readXXX`. It is normally only useful when the connection has been opened for nonblocking I/O with the `O_NONBLOCK` flag, especially when I/O is optimized with the help of the `select(2)` call.

  Usually some request must have been sent before there is anything to read. When the method is `GET_METHOD` and the `O_NONBLOCK` has *not* been specified, the `openXXX` will sent the request and the browser can immediately call `readXXX` to wait for data. There is no need for `peekXXX` in this situation.

  However, when the `O_NONBLOCK` flag is given to `openXXX`, the `openXXX` function should return immediately and it might not have succeeded in sending the request. In that case the `peekXXX` function should be called repeatedly, until it returns non-zero.

> **Note:** *To optimize the non-blocking I/ O,* `peekXXX` *can be used in combination with* `select(2)` *to prevent unnecessary calls to* `peekXXX`. *The function only needs to be called when the file descriptor is ready for* writing *(not reading!).*
>
> `select(2)` *can also be used to prevent unnecessary calls to* `readXXX`, *but only* after `peekXXX` *has returned a positive number at least once.*

> **Note:** `peekXXX` *should not return -1 with an error code of* `EAGAIN`. *It should return 0 instead.*

- `int writeXXX(int fd, const char *buf, size_t nbytes)`

  Returns the number of characters actually sent, or -1 in case of error. The browser should repeat the call with any character that remained unwritten. The `fd` must be one that is returned by a call to `openXXX` with a method of `PUT_METHOD` or `POST_METHOD`.

- `Bool closeXXX(int fd)`

  Closes the connection. If the return code is `FALSE`, *errno* contains the **error code**.

- `Bool deleteXXX(const char *url)`

  Tries to delete the indicated document. If the return code is `FALSE`, *errno* contains the **error code**.

- `Bool infoXXX(int fd, W3ADocumentInfo *buf)`

  Returns information about the retrieved document. Returns `TRUE` on success, `FALSE` on error. `W3ADocumentInfo` is a struct defined in `w3a.h`.

  The function will return at least the MIME type of the document. Other information may be available, depending on the protocol and the document.

  If the connection has been opened without the `O_NONBLOCK` flag, the function will not return until it knows the requested information, even if that means reading additional data from the document. (The data will be buffered, so that it can still be returned by `readXXX`).

  On the other hand, if the connection has been opened for non-blocking I/O with `O_NONBLOCK`, the `infoXXX` function will return immediately, whether or not the information is available. If not, it returns `FALSE` and sets `errno` to `EAGAIN`. The function can be called repeatedly until it succeeds. It will buffer any data that it must skip in order to reach the desired information.

> **Note:** *When used in combination with* `select(2)`*, use the read mask to see when it is useful to call* `infoXXX` *again.*

The browser uses these functions subject to the obvious constraints: `initXXX` must be called exactly once, prior to the first call to `openXXX`. The functions `readXXX`, `writeXXX`, `filenoXXX` and `infoXXX` can only be called with an object that is returned by `openXXX`. After a call to `closeXXX` no further calls to `readXXX`, `writeXXX`, or `infoXXX` are possible for the same object.

Question: *How about URNs? I guess resolving URNs to URLs will be an (exported) function of the browser.* **See** `W3AresolveURN()`.
