ApplicationCache Error Detail Proposal

go/appcacheerrorsspec

The WHATWG HTML Living Standard's Offline Web Applications section steps for "Downloading or updating an application cache" defines errors raising behavior via the phrasing "*create a* ***task*** *to* ***fire a simple event*** *that is cancelable named* ***error***". This occurs 3 times in the application cache download process, and twice in the "*cache failure steps*" which are referenced 5 times in the process - so there are eight distinct sources for error events in the process.

To add additional information to the error, each of these sources must be augmented with the additional data to include with the error event.

Proposed Spec Addition

IDL:

```
enum ApplicationCacheErrorReason {
  "manifest",         // manifest fetch failure (other than obsolete)
  "signature",        // parser for manifest failed when checking signature
  "resource",         // resource fetch failure
  "changed",          // second manifest was not byte-for-byte identical
  "abort",            // abort() was called
  "quota",            // user agent failure due to quota limitations
  "policy",           // user agent failure due to policy limitations
  "unknown"           // user agent failure for other reason (I/O, DB, etc)
};

[Constructor(DOMString type, optional ApplicationCacheErrorEventInit eventInitDict),
Exposed=Window]
interface ApplicationCacheErrorEvent : Event {
  readonly attribute ApplicationCacheErrorReason reason;
  readonly attribute DOMString url;            // Empty string unless a fetch
failure
  readonly attribute unsigned short status;    // 0 means no status returned by
server
  readonly attribute DOMString message;        // User agent-specific
};
```

To **create an application cache error task** with *target, reason*, optional *url* and optional *status*, run these steps:

1. Let *event* be a new trusted **ApplicationCacheErrorEvent** object that does not bubble but is cancelable, and which has the event name **error**.
2. Initialize *event*'s **reason** attribute to *reason*

3. If url was given, initialize *event*'s **url** attribute to *url*; otherwise, initialize *event*'s **url** attribute to the empty string
4. If status was given, initialize event's **status** attribute to *status*; otherwise, initialize event's **status** attribute to 0.
5. Initialize *event*'s **message** attribute to a user agent-defined value.
6. Create and return a task to dispatch *event* at *target*. The default action of this event must be, if the user agent shows caching progress, the display of some sort of user interface indicating to the user that the user agent failed to save the application for offline use.

If the user agent allows resources to be fetched from a different origin than the manifest and the resource fetch fails, *event*'s **status** MUST be reported as 0 and the *event*'s **message** MUST NOT include additional details about the failure.

Proposed Spec Changes

#1: fetching the manifest fails due to 404 or 410 or equivalent:

> Step 5: *If fetching the manifest fails due to a 404 or 410 response or equivalent, then run these* substeps:
>
> > Substep 4: *For each entry in cache group's list of pending master entries,* ***create an application cache error task*** *with the ApplicationCache singleton of the Document for this entry as target, and "manifest" as reason and the server response code (if any) as status….*

#2: upgrade attempt, and resource download failed

> Step 7: *If this is an upgrade attempt and the newly downloaded manifest is byte-for-byte identical to the manifest found in the newest application cache in cache group, or the server reported it as "304 Not Modified" or equivalent, then run these substeps:*
>
> > Substep 3: *For each entry in cache group's list of pending master entries, wait for the resource for this entry to have either completely downloaded or failed.*
> >
> > *…*
> >
> > *If the download failed (e.g. the server returns a 4xx or 5xx response or equivalent, or there is a DNS error, the connection times out, or the user cancels the download), or if the resource is labeled with the "no-store" cache directive, then* ***create an application cache error task*** *with the ApplicationCache singleton of the Document for this entry, if there still is one, as target, and "resource" as reason, the resource url as url, and the server response code (if any) as status.*

#3: otherwise, and resource download failed

> Step 23: *For each entry in cache group's list of pending master entries, wait for the resource for this entry to have either completely downloaded or failed.*

If the download failed (e.g. the server returns a 4xx or 5xx response or equivalent, or there is a DNS error, the connection times out, or the user cancels the download), or if the resource is labeled with the "no-store" cache directive, then run these substeps:

Substep 2: **Create an application cache error task** with the *ApplicationCache singleton of the Document for this entry, if there still is one, as target, "resource" as reason resource url as url, and the server response code (if any) as status,* and *queue it as a post-load task.*

#4: fetching manifest fails due to non-404/410 (i.e. 4xx, 5xx, DNS, timeout, user cancel, parse failure, etc)

Step 6: *Otherwise, if fetching the manifest fails in some other way (e.g. the server returns another 4xx or 5xx response or equivalent, or there is a DNS error, or the connection times out, or the user cancels the download, or the parser for manifests fails when checking the magic signature), or if the server returned a redirect, then* ***run the cache failure steps*** *with "signature" as reason if the parser failed, or "manifest" as reason otherwise**,** and the server response code (if any) as status*

#5: explicit abort() call from script

Step 17: *For each URL in file list, run the following steps… If, while running these steps, the ApplicationCache object's abort() method sends a signal to this instance of the application cache download process algorithm, then* ***run the cache failure steps*** *with "abort" as reason instead.*

#6: resource fetch fails (4xx, 5xx, DNS, timeout, user cancel, etc.) or redirect and resource is explicit entry or fallback entry

Step 17: *For each URL in file list, run the following steps….*

Substep 4: *If the previous step fails (e.g. the server returns a 4xx or 5xx response or equivalent, or there is a DNS error, or the connection times out, or the user cancels the download), or if the server returned a redirect, or if the resource is labeled with the "no-store" cache directive, then...*

If the URL being processed was flagged as an "explicit entry" or a "fallback entry" … ***Run the cache failure steps*** *with "resource" as reason resource url as url, and the server response code (if any) as status*

#7: user agent failure (e.g. quota)

Step 17: *For each URL in file list, run the following steps...*

Substep 5: *If the user agent is not able to store the resource (e.g. because of quota restrictions)... the user agent must* ***run the cache failure steps*** *with "quota", "policy", or "unknown" as reason...*

#8: manifest changed on re-fetch

Step 25: *If the previous step failed for any reason, or if the fetching attempt involved a redirect, or if second manifest and manifest are not byte-for-byte identical, then schedule* *a rerun of the entire algorithm with the same parameters after a short delay, and* ***run the cache failure steps*** *with “changed” as reason if the manifests were not byte-for-byte identical, or “manifest” as reason and the server response code (if any) as status otherwise.*

The **cache failure steps** with *reason*, optional *url* and optional *status*, are as follows:

…

Step 2: *For each entry in cache group's list of pending master entries…*

...

Substep 3: ***Create an application cache error task*** *with the ApplicationCache singleton of the Document for this entry, if there still is one, as target, reason as reason, url as url (if given) and status as status (if given)...*

...

Step 3: *For each cache host still associated with an application cache in cache group,* ***create an application cache error task*** *with the ApplicationCache singleton of the cache host as target, and reason as reason.*

...