

**NAME**

**fido\_assert\_set\_authdata**, **fido\_assert\_set\_authdata\_raw**, **fido\_assert\_set\_clientdata**,  
**fido\_assert\_set\_clientdata\_hash**, **fido\_assert\_set\_count**, **fido\_assert\_set\_extensions**,  
**fido\_assert\_set\_hmac\_salt**, **fido\_assert\_set\_hmac\_secret**, **fido\_assert\_set\_up**, **fido\_assert\_set\_uv**,  
**fido\_assert\_set\_rp**, **fido\_assert\_set\_sig** - set parameters of a FIDO2 assertion

**SYNOPSIS**

```
#include <fido.h>
```

```
typedef enum {
```

```
    FIDO_OPT_OMIT = 0, /* use authenticator's default */
```

```
    FIDO_OPT_FALSE, /* explicitly set option to false */
```

```
    FIDO_OPT_TRUE, /* explicitly set option to true */
```

```
} fido_opt_t;
```

```
int
```

```
fido_assert_set_authdata(fido_assert_t *assert, size_t idx, const unsigned char *ptr, size_t len);
```

```
int
```

```
fido_assert_set_authdata_raw(fido_assert_t *assert, size_t idx, const unsigned char *ptr, size_t len);
```

```
int
```

```
fido_assert_set_clientdata(fido_assert_t *assert, const unsigned char *ptr, size_t len);
```

```
int
```

```
fido_assert_set_clientdata_hash(fido_assert_t *assert, const unsigned char *ptr, size_t len);
```

```
int
```

```
fido_assert_set_count(fido_assert_t *assert, size_t n);
```

```
int
```

```
fido_assert_set_extensions(fido_assert_t *assert, int flags);
```

```
int
```

```
fido_assert_set_hmac_salt(fido_assert_t *assert, const unsigned char *ptr, size_t len);
```

```
int
```

```
fido_assert_set_hmac_secret(fido_assert_t *assert, size_t idx, const unsigned char *ptr, size_t len);
```

```
int
```

```
fido_assert_set_up(fido_assert_t *assert, fido_opt_t up);
```

*int*

**fido\_assert\_set\_uv**(*fido\_assert\_t* \**assert*, *fido\_opt\_t* *uv*);

*int*

**fido\_assert\_set\_rp**(*fido\_assert\_t* \**assert*, *const char* \**id*);

*int*

**fido\_assert\_set\_sig**(*fido\_assert\_t* \**assert*, *size\_t* *idx*, *const unsigned char* \**ptr*, *size\_t* *len*);

## DESCRIPTION

The **fido\_assert\_set\_authdata** set of functions define the various parameters of a FIDO2 assertion, allowing a *fido\_assert\_t* type to be prepared for a subsequent call to **fido\_dev\_get\_assert(3)** or **fido\_assert\_verify(3)**. For the complete specification of a FIDO2 assertion and the format of its constituent parts, please refer to the Web Authentication (webauthn) standard.

The **fido\_assert\_set\_count()** function sets the number of assertion statements in *assert* to *n*.

The **fido\_assert\_set\_authdata()** and **fido\_assert\_set\_sig()** functions set the authenticator data and signature parts of the statement with index *idx* of *assert* to *ptr*, where *ptr* points to *len* bytes. A copy of *ptr* is made, and no references to the passed pointer are kept. Please note that the first assertion statement of *assert* has an *idx* of 0. The authenticator data passed to **fido\_assert\_set\_authdata()** must be a CBOR-encoded byte string, as obtained from **fido\_assert\_authdata\_ptr()**. Alternatively, a raw binary blob may be passed to **fido\_assert\_set\_authdata\_raw()**.

The **fido\_assert\_set\_clientdata\_hash()** function sets the client data hash of *assert* to *ptr*, where *ptr* points to *len* bytes. A copy of *ptr* is made, and no references to the passed pointer are kept.

The **fido\_assert\_set\_clientdata()** function allows an application to set the client data hash of *assert* by specifying the assertion's unhashed client data. This is required by Windows Hello, which calculates the client data hash internally. For compatibility with Windows Hello, applications should use **fido\_assert\_set\_clientdata()** instead of **fido\_assert\_set\_clientdata\_hash()**.

The **fido\_assert\_set\_rp()** function sets the relying party *id* of *assert*, where *id* is a NUL-terminated UTF-8 string. The content of *id* is copied, and no references to the passed pointer are kept.

The **fido\_assert\_set\_extensions()** function sets the extensions of *assert* to the bitmask *flags*. At the moment, only the FIDO\_EXT\_CRED\_BLOB, FIDO\_EXT\_HMAC\_SECRET, and FIDO\_EXT\_LARGELOB\_KEY extensions are supported. If *flags* is zero, the extensions of *assert* are cleared.

The **fid0\_assert\_set\_hmac\_salt()** and **fid0\_assert\_set\_hmac\_secret()** functions set the hmac-salt and hmac-secret parts of *assert* to *ptr*, where *ptr* points to *len* bytes. A copy of *ptr* is made, and no references to the passed pointer are kept. The HMAC Secret (hmac-secret) Extension is a CTAP 2.0 extension. Note that the resulting hmac-secret varies according to whether user verification was performed by the authenticator. The **fid0\_assert\_set\_hmac\_secret()** function is normally only useful when writing tests.

The **fid0\_assert\_set\_up()** and **fid0\_assert\_set\_uv()** functions set the *up* (user presence) and *uv* (user verification) attributes of *assert*. Both are FIDO\_OPT\_OMIT by default, allowing the authenticator to use its default settings.

Use of the **fid0\_assert\_set\_authdata** set of functions may happen in two distinct situations: when asking a FIDO2 device to produce a series of assertion statements, prior to **fid0\_dev\_get\_assert(3)** (i.e, in the context of a FIDO2 client), or when verifying assertion statements using **fid0\_assert\_verify(3)** (i.e, in the context of a FIDO2 server).

For a complete description of the generation of a FIDO2 assertion and its verification, please refer to the FIDO2 specification. An example of how to use the **fid0\_assert\_set\_authdata** set of functions can be found in the *examples/assert.c* file shipped with *libfido2*.

## RETURN VALUES

The **fid0\_assert\_set\_authdata** functions return FIDO\_OK on success. The error codes returned by the **fid0\_assert\_set\_authdata** set of functions are defined in *<fido/err.h>*.

## SEE ALSO

**fid0\_assert\_allow\_cred(3)**, **fid0\_assert\_verify(3)**, **fid0\_dev\_get\_assert(3)**