

**NAME**

EC\_KEY\_new, EC\_KEY\_get\_flags, EC\_KEY\_set\_flags, EC\_KEY\_clear\_flags,  
 EC\_KEY\_new\_by\_curve\_name, EC\_KEY\_free, EC\_KEY\_copy, EC\_KEY\_dup, EC\_KEY\_up\_ref,  
 EC\_KEY\_get0\_group, EC\_KEY\_set\_group, EC\_KEY\_get0\_private\_key,  
 EC\_KEY\_set\_private\_key, EC\_KEY\_get0\_public\_key, EC\_KEY\_set\_public\_key,  
 EC\_KEY\_get\_enc\_flags, EC\_KEY\_set\_enc\_flags, EC\_KEY\_get\_conv\_form,  
 EC\_KEY\_set\_conv\_form, EC\_KEY\_get\_key\_method\_data, EC\_KEY\_insert\_key\_method\_data,  
 EC\_KEY\_set\_asn1\_flag, EC\_KEY\_precompute\_mult, EC\_KEY\_generate\_key,  
 EC\_KEY\_check\_key, EC\_KEY\_set\_public\_key\_affine\_coordinates - Functions for creating,  
 destroying and manipulating EC\_KEY objects.

**SYNOPSIS**

```
#include <openssl/ec.h>
#include <openssl/bn.h>

EC_KEY *EC_KEY_new(void);
int EC_KEY_get_flags(const EC_KEY *key);
void EC_KEY_set_flags(EC_KEY *key, int flags);
void EC_KEY_clear_flags(EC_KEY *key, int flags);
EC_KEY *EC_KEY_new_by_curve_name(int nid);
void EC_KEY_free(EC_KEY *key);
EC_KEY *EC_KEY_copy(EC_KEY *dst, const EC_KEY *src);
EC_KEY *EC_KEY_dup(const EC_KEY *src);
int EC_KEY_up_ref(EC_KEY *key);
const EC_GROUP *EC_KEY_get0_group(const EC_KEY *key);
int EC_KEY_set_group(EC_KEY *key, const EC_GROUP *group);
const BIGNUM *EC_KEY_get0_private_key(const EC_KEY *key);
int EC_KEY_set_private_key(EC_KEY *key, const BIGNUM *prv);
const EC_POINT *EC_KEY_get0_public_key(const EC_KEY *key);
int EC_KEY_set_public_key(EC_KEY *key, const EC_POINT *pub);
point_conversion_form_t EC_KEY_get_conv_form(const EC_KEY *key);
void EC_KEY_set_conv_form(EC_KEY *eckey, point_conversion_form_t cform);
void *EC_KEY_get_key_method_data(EC_KEY *key,
void *(*dup_func)(void *), void (*free_func)(void *), void (*clear_free_func)(void *));
void EC_KEY_insert_key_method_data(EC_KEY *key, void *data,
void *(*dup_func)(void *), void (*free_func)(void *), void (*clear_free_func)(void *));
void EC_KEY_set_asn1_flag(EC_KEY *eckey, int asn1_flag);
int EC_KEY_precompute_mult(EC_KEY *key, BN_CTX *ctx);
int EC_KEY_generate_key(EC_KEY *key);
int EC_KEY_check_key(const EC_KEY *key);
int EC_KEY_set_public_key_affine_coordinates(EC_KEY *key, BIGNUM *x, BIGNUM *y);
```

**DESCRIPTION**

An EC\_KEY represents a public key and (optionally) an associated private key. A new EC\_KEY (with no associated curve) can be constructed by calling EC\_KEY\_new. The reference count for the newly created EC\_KEY is initially set to 1. A curve can be associated with the EC\_KEY by calling EC\_KEY\_set\_group.

Alternatively a new EC\_KEY can be constructed by calling EC\_KEY\_new\_by\_curve\_name and supplying the nid of the associated curve. Refer to [EC\\_GROUP\\_new\(3\)](#) for a description of curve names. This function simply wraps calls to EC\_KEY\_new and EC\_GROUP\_new\_by\_curve\_name.

Calling EC\_KEY\_free decrements the reference count for the EC\_KEY object, and if it has dropped to zero then frees the memory associated with it.

EC\_KEY\_copy copies the contents of the EC\_KEY in **src** into **dest**.

EC\_KEY\_dup creates a new EC\_KEY object and copies **ec\_key** into it.

`EC_KEY_up_ref` increments the reference count associated with the `EC_KEY` object.

`EC_KEY_generate_key` generates a new public and private key for the supplied `eckey` object. `eckey` must have an `EC_GROUP` object associated with it before calling this function. The private key is a random integer ( $0 < \text{priv\_key} < \text{order}$ , where `order` is the order of the `EC_GROUP` object). The public key is an `EC_POINT` on the curve calculated by multiplying the generator for the curve by the private key.

`EC_KEY_check_key` performs various sanity checks on the `EC_KEY` object to confirm that it is valid.

`EC_KEY_set_public_key_affine_coordinates` sets the public key for `key` based on its affine coordinates, i.e. it constructs an `EC_POINT` object based on the supplied `x` and `y` values and sets the public key to be this `EC_POINT`. It will also perform certain sanity checks on the key to confirm that it is valid.

The functions `EC_KEY_get0_group`, `EC_KEY_set_group`, `EC_KEY_get0_private_key`, `EC_KEY_set_private_key`, `EC_KEY_get0_public_key`, and `EC_KEY_set_public_key` get and set the `EC_GROUP` object, the private key and the `EC_POINT` public key for the `key` respectively.

The functions `EC_KEY_get_conv_form` and `EC_KEY_set_conv_form` get and set the `point_conversion_form` for the `key`. For a description of `point_conversion_forms` please refer to [EC\\_POINT\\_new\(3\)](#).

`EC_KEY_insert_key_method_data` and `EC_KEY_get_key_method_data` enable the caller to associate arbitrary additional data specific to the elliptic curve scheme being used with the `EC_KEY` object. This data is treated as a “black box” by the `ec` library. The data to be stored by `EC_KEY_insert_key_method_data` is provided in the `data` parameter, which must have associated functions for duplicating, freeing and “clear freeing” the data item. If a subsequent `EC_KEY_get_key_method_data` call is issued, the functions for duplicating, freeing and “clear freeing” the data item must be provided again, and they must be the same as they were when the data item was inserted.

`EC_KEY_set_flags` sets the flags in the `flags` parameter on the `EC_KEY` object. Any flags that are already set are left set. The currently defined standard flags are `EC_FLAG_NON_FIPS_ALLOW` and `EC_FLAG_FIPS_CHECKED`. In addition there is the flag `EC_FLAG_COFACTOR_ECDH` which is specific to ECDH and is defined in `ecdh.h`. `EC_KEY_get_flags` returns the current flags that are set for this `EC_KEY`. `EC_KEY_clear_flags` clears the flags indicated by the `flags` parameter. All other flags are left in their existing state.

`EC_KEY_set_asn1_flag` sets the `asn1_flag` on the underlying `EC_GROUP` object (if set). Refer to [EC\\_GROUP\\_copy\(3\)](#) for further information on the `asn1_flag`.

`EC_KEY_precompute_mult` stores multiples of the underlying `EC_GROUP` generator for faster point multiplication. See also [EC\\_POINT\\_add\(3\)](#).

## RETURN VALUES

`EC_KEY_new`, `EC_KEY_new_by_curve_name` and `EC_KEY_dup` return a pointer to the newly created `EC_KEY` object, or `NULL` on error.

`EC_KEY_get_flags` returns the flags associated with the `EC_KEY` object as an integer.

`EC_KEY_copy` returns a pointer to the destination key, or `NULL` on error.

`EC_KEY_up_ref`, `EC_KEY_set_group`, `EC_KEY_set_private_key`, `EC_KEY_set_public_key`, `EC_KEY_precompute_mult`, `EC_KEY_generate_key`, `EC_KEY_check_key` and `EC_KEY_set_public_key_affine_coordinates` return 1 on success or 0 on error.

`EC_KEY_get0_group` returns the `EC_GROUP` associated with the `EC_KEY`.

`EC_KEY_get0_private_key` returns the private key associated with the `EC_KEY`.

`EC_KEY_get_conv_form` return the `point_conversion_form` for the `EC_KEY`.

**SEE ALSO**

*crypto(3)*, *ec(3)*, *EC\_GROUP\_new(3)*, *EC\_GROUP\_copy(3)*, *EC\_POINT\_new(3)*,  
*EC\_POINT\_add(3)*, *EC\_GFp\_simple\_method(3)*, *d2i\_ECPKParameters(3)*