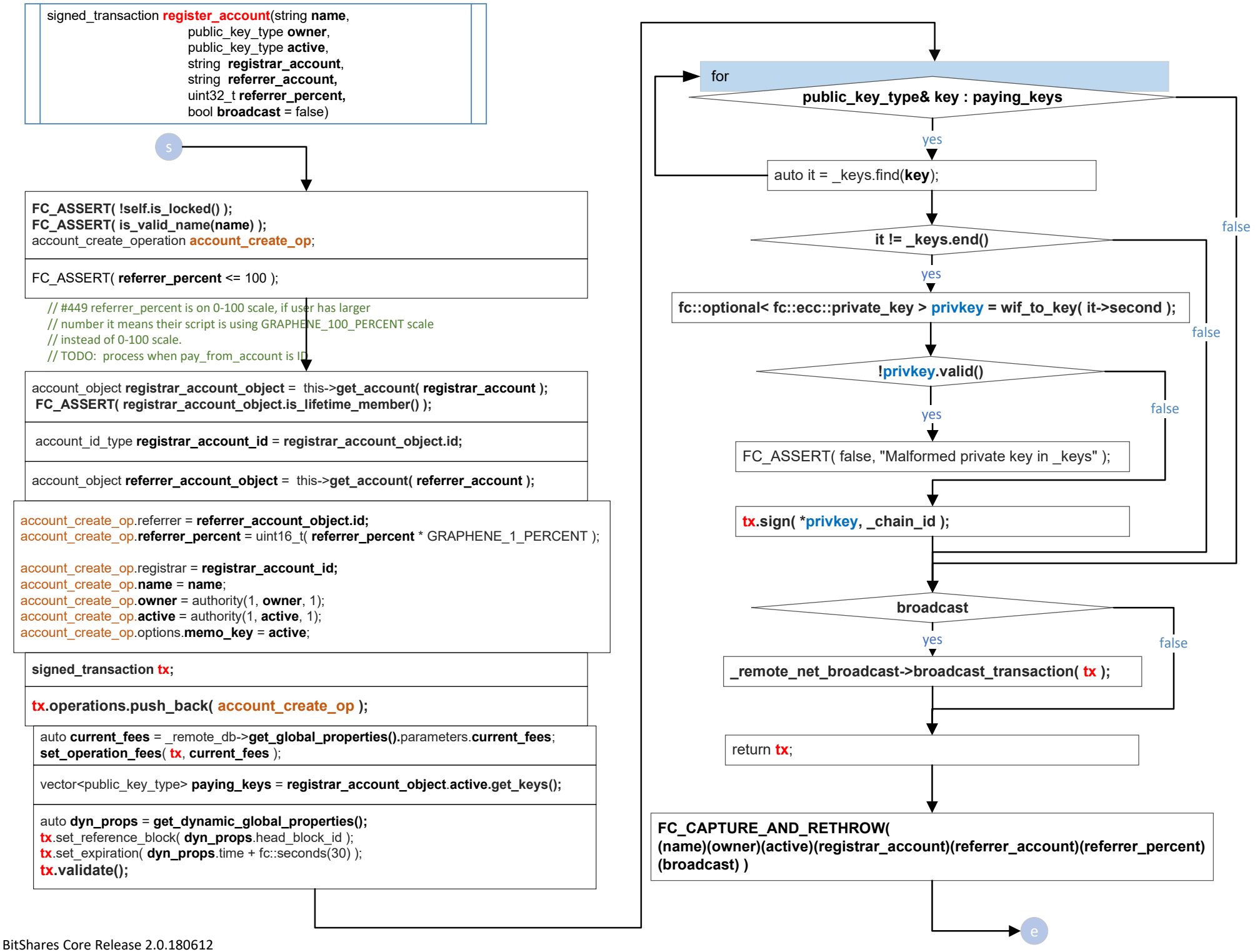
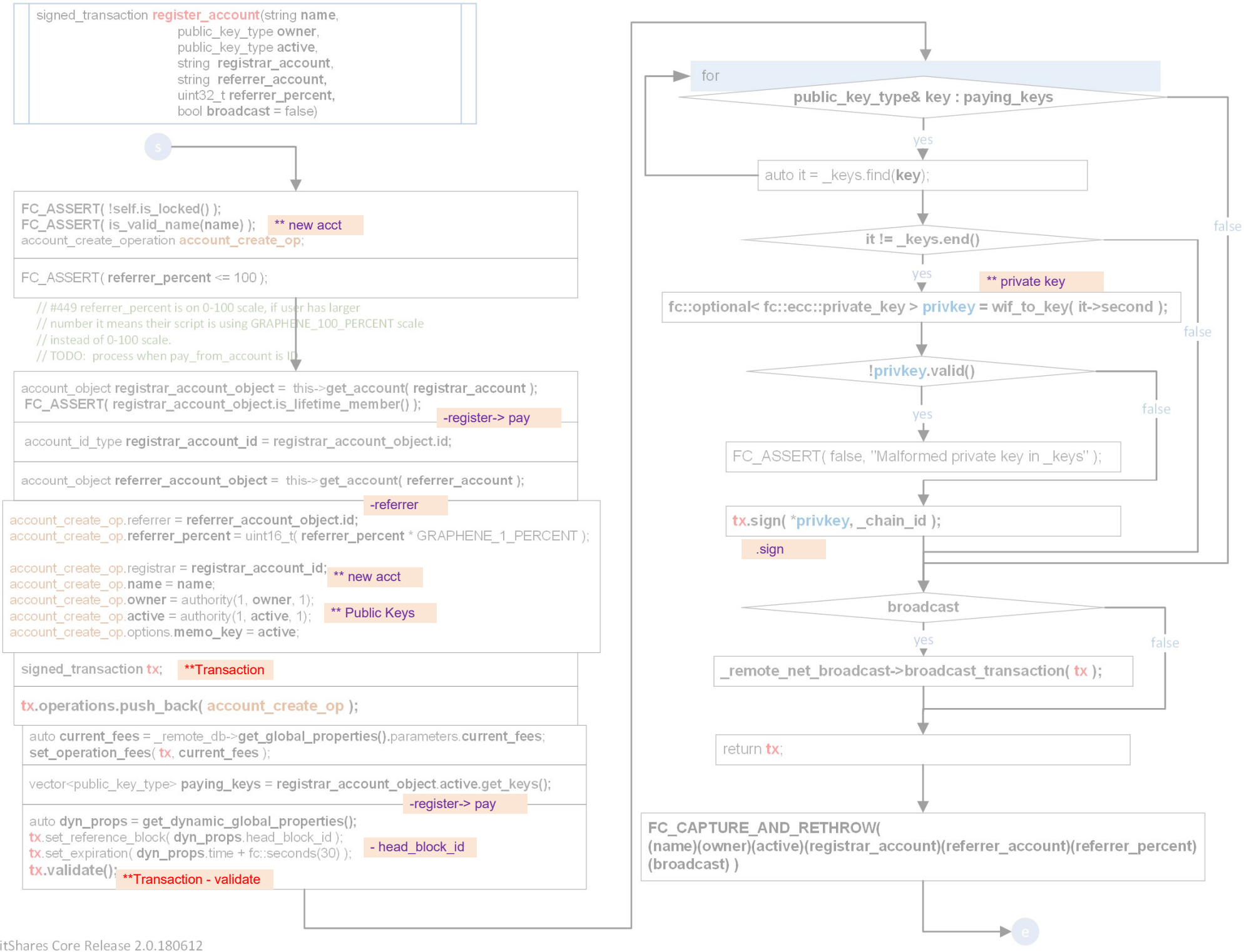

signed_transaction register_account(string name,
public_key_type owner,
public_key_type active,
string registrar_account,
string referrer_account,
uint32_t referrer_percent,
bool broadcast = false)
s
FC_ASSERT(!self.is_locked());
FC_ASSERT(is_valid_name(name));
account_create_operation account_create_op;
FC_ASSERT(referrer_percent <= 100);
// #449 referrer_percent is on 0-100 scale, if user has larger
// number it means their script is using GRAPHENE_100_PERCENT scale
// instead of 0-100 scale.
// TODO: process when pay_from_account is ID
account_object registrar_account_object = this->get_account(registrar_account);
FC_ASSERT(registrar_account_object.is_lifetime_member());
account_id_type registrar_account_id = registrar_account_object.id;
account_object referrer_account_object = this->get_account(referrer_account);
account_create_op.referrer = referrer_account_object.id;
account_create_op.referrer_percent = uint16_t(referrer_percent * GRAPHENE_1_PERCENT);
account_create_op.registrar = registrar_account_id;
account_create_op.name = name;
account_create_op.owner = authority(1, owner, 1);
account_create_op.active = authority(1, active, 1);
account_create_op.options.memo_key = active;
signed_transaction tx;
tx.operations.push_back(account_create_op);
auto current_fees = _remote_db->get_global_properties().parameters.current_fees;
set_operation_fees(tx, current_fees);
vector<public_key_type> paying_keys = registrar_account_object.active.get_keys();
auto dyn_props = get_dynamic_global_properties();
tx.set_reference_block(dyn_props.head_block_id);
tx.set_expiration(dyn_props.time + fc::seconds(30));
tx.validate();
for
public_key_type& key : paying_keys
yes
auto it = _keys.find(key);
it != _keys.end()
false
yes
fc::optional< fc::ecc::private_key > privkey = wif_to_key(it->second);
false
!privkey.valid()
false
yes
FC_ASSERT(false, "Malformed private key in _keys");
tx.sign(*privkey, _chain_id);
broadcast
false
yes
_remote_net_broadcast->broadcast_transaction(tx);
return tx;
FC_CAPTURE_AND_RETHROW(
(name)(owner)(active)(registrar_account)(referrer_account)(referrer_percent)
(broadcast))
e


signed_transaction register_account(string name,
public_key_type owner,
public_key_type active,
string registrar_account,
string referrer_account,
uint32_t referrer_percent,
bool broadcast = false)
s
FC_ASSERT(!self.is_locked());
FC_ASSERT(is_valid_name(name));
** new acct
account_create_operation account_create_op;
FC_ASSERT(referrer_percent <= 100);
// #449 referrer_percent is on 0-100 scale, if user has larger
// number it means their script is using GRAPHENE_100_PERCENT scale
// instead of 0-100 scale.
// TODO: process when pay_from_account is ID
account_object registrar_account_object = this->get_account(registrar_account);
FC_ASSERT(registrar_account_object.is_lifetime_member());
-register-> pay
account_id_type registrar_account_id = registrar_account_object.id;
account_object referrer_account_object = this->get_account(referrer_account);
-referrer
account_create_op.referrer = referrer_account_object.id;
account_create_op.referrer_percent = uint16_t(referrer_percent * GRAPHENE_1_PERCENT);
account_create_op.registrar = registrar_account_id;
** new acct
account_create_op.name = name;
account_create_op.owner = authority(1, owner, 1);
account_create_op.active = authority(1, active, 1);
** Public Keys
account_create_op.options.memo_key = active;
signed_transaction tx;
**Transaction
tx.operations.push_back(account_create_op);
auto current_fees = _remote_db->get_global_properties().parameters.current_fees;
set_operation_fees(tx, current_fees);
vector<public_key_type> paying_keys = registrar_account_object.active.get_keys();
-register-> pay
auto dyn_props = get_dynamic_global_properties();
tx.set_reference_block(dyn_props.head_block_id);
- head_block_id
tx.set_expiration(dyn_props.time + fc::seconds(30));
tx.validate();
**Transaction - validate
for
public_key_type& key : paying_keys
yes
auto it = _keys.find(key);
it != _keys.end()
false
yes
** private key
fc::optional< fc::ecc::private_key > privkey = wif_to_key(it->second);
false
!privkey.valid()
false
yes
FC_ASSERT(false, "Malformed private key in _keys");
tx.sign(*privkey, _chain_id);
.sign
broadcast
false
yes
_remote_net_broadcast->broadcast_transaction(tx);
return tx;
FC_CAPTURE_AND_RETHROW(
(name)(owner)(active)(registrar_account)(referrer_account)(referrer_percent)
(broadcast))
e