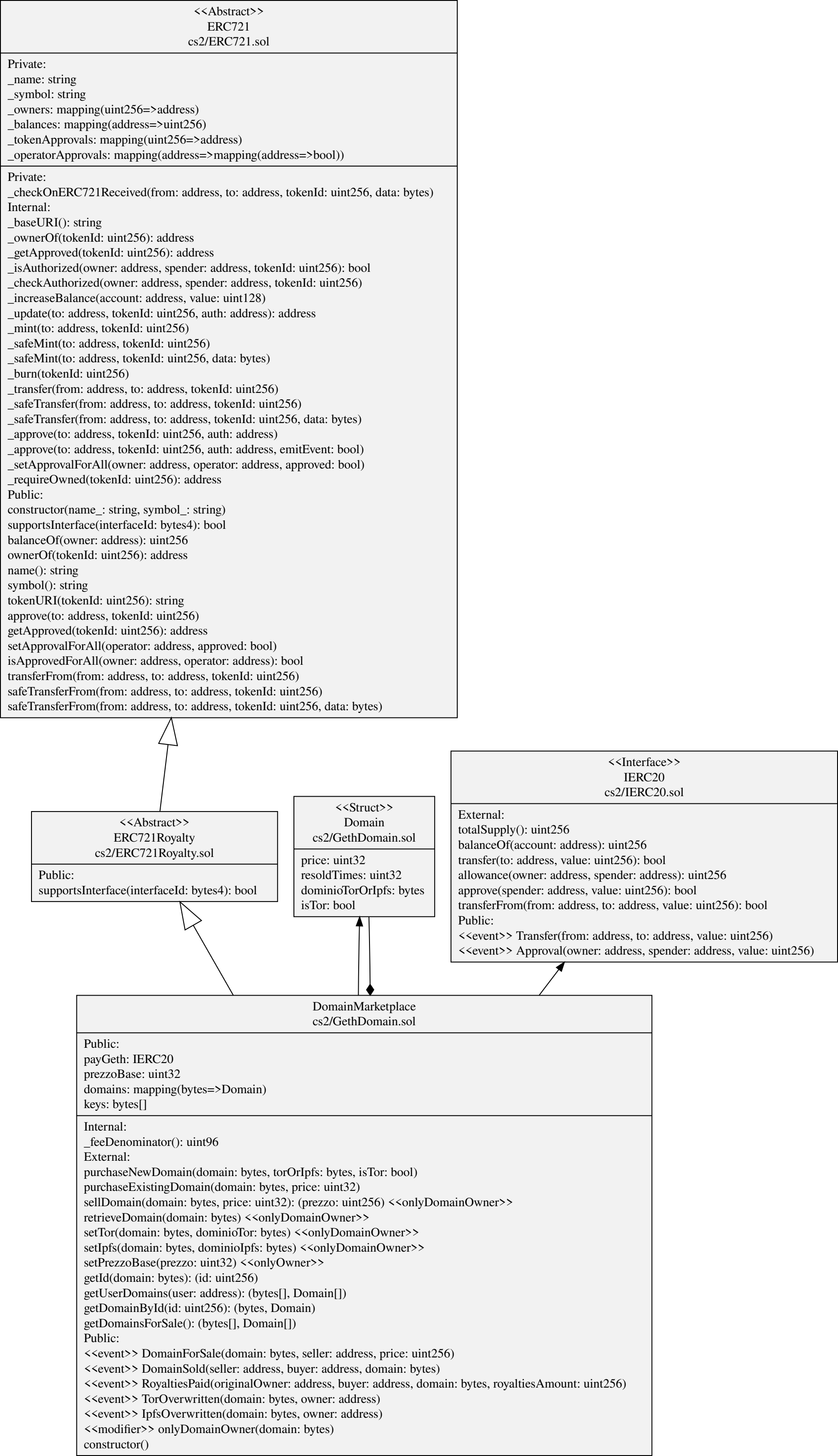
<<Abstract>>
ERC721
cs2/ERC721.sol
Private:
_name: string
_symbol: string
_owners: mapping(uint256=>address)
_balances: mapping(address=>uint256)
_tokenApprovals: mapping(uint256=>address)
_operatorApprovals: mapping(address=>mapping(address=>bool))
Private:
_checkOnERC721Received(from: address, to: address, tokenId: uint256, data: bytes)
Internal:
_baseURI(): string
_ownerOf(tokenId: uint256): address
_getApproved(tokenId: uint256): address
_isAuthorized(owner: address, spender: address, tokenId: uint256): bool
_checkAuthorized(owner: address, spender: address, tokenId: uint256)
_increaseBalance(account: address, value: uint128)
_update(to: address, tokenId: uint256, auth: address): address
_mint(to: address, tokenId: uint256)
_safeMint(to: address, tokenId: uint256)
_safeMint(to: address, tokenId: uint256, data: bytes)
_burn(tokenId: uint256)
_transfer(from: address, to: address, tokenId: uint256)
_safeTransfer(from: address, to: address, tokenId: uint256)
_safeTransfer(from: address, to: address, tokenId: uint256, data: bytes)
_approve(to: address, tokenId: uint256, auth: address)
_approve(to: address, tokenId: uint256, auth: address, emitEvent: bool)
_setApprovalForAll(owner: address, operator: address, approved: bool)
_requireOwned(tokenId: uint256): address
Public:
constructor(name_: string, symbol_: string)
supportsInterface(interfaceId: bytes4): bool
balanceOf(owner: address): uint256
ownerOf(tokenId: uint256): address
name(): string
symbol(): string
tokenURI(tokenId: uint256): string
approve(to: address, tokenId: uint256)
getApproved(tokenId: uint256): address
setApprovalForAll(operator: address, approved: bool)
isApprovedForAll(owner: address, operator: address): bool
transferFrom(from: address, to: address, tokenId: uint256)
safeTransferFrom(from: address, to: address, tokenId: uint256)
safeTransferFrom(from: address, to: address, tokenId: uint256, data: bytes)
<<Abstract>>
ERC721Royalty
cs2/ERC721Royalty.sol
Public:
supportsInterface(interfaceId: bytes4): bool
<<Struct>>
Domain
cs2/GethDomain.sol
price: uint32
resoldTimes: uint32
dominioTorOrIpfs: bytes
isTor: bool
<<Interface>>
IERC20
cs2/IERC20.sol
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, value: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, value: uint256): bool
transferFrom(from: address, to: address, value: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
DomainMarketplace
cs2/GethDomain.sol
Public:
payGeth: IERC20
prezzoBase: uint32
domains: mapping(bytes=>Domain)
keys: bytes[]
Internal:
_feeDenominator(): uint96
External:
purchaseNewDomain(domain: bytes, torOrIpfs: bytes, isTor: bool)
purchaseExistingDomain(domain: bytes, price: uint32)
sellDomain(domain: bytes, price: uint32): (prezzo: uint256) <<onlyDomainOwner>>
retrieveDomain(domain: bytes) <<onlyDomainOwner>>
setTor(domain: bytes, dominioTor: bytes) <<onlyDomainOwner>>
setIpfs(domain: bytes, dominioIpfs: bytes) <<onlyDomainOwner>>
setPrezzoBase(prezzo: uint32) <<onlyOwner>>
getId(domain: bytes): (id: uint256)
getUserDomains(user: address): (bytes[], Domain[])
getDomainById(id: uint256): (bytes, Domain)
getDomainsForSale(): (bytes[], Domain[])
Public:
<<event>> DomainForSale(domain: bytes, seller: address, price: uint256)
<<event>> DomainSold(seller: address, buyer: address, domain: bytes)
<<event>> RoyaltiesPaid(originalOwner: address, buyer: address, domain: bytes, royaltiesAmount: uint256)
<<event>> TorOverwritten(domain: bytes, owner: address)
<<event>> IpfsOverwritten(domain: bytes, owner: address)
<<modifier>> onlyDomainOwner(domain: bytes)
constructor()