

```

1 // SPDX-License-Identifier: MIT
2 // OpenZeppelin Contracts (last updated v4.5.0) (proxy/ERC1967/ERC1967Upgrade.sol)
3
4 pragma solidity ^0.8.2;
5
6 import "../beacon/IBeacon.sol";
7 import "../interfaces/draft-IERC1822.sol";
8 import "../utils/Address.sol";
9 import "../utils/StorageSlot.sol";
10
11 /**
12  * @dev This abstract contract provides getters and event emitting update functions
13  * for
14  * https://eips.ethereum.org/EIPS/eip-1967[EIP1967] slots.
15  *
16  * _Available since v4.1._
17  *
18  * @custom:oz-upgrades-unsafe-allow delegatecall
19  */
20 abstract contract ERC1967Upgrade {
21     // This is the keccak-256 hash of "eip1967.proxy.rollback" subtracted by 1
22     bytes32 private constant _ROLLBACK_SLOT =
23         0x4910fdafa16fed3260ed0e7147f7cc6da1a60208b5b9406d12a635614ffd9143;
24
25     /**
26      * @dev Storage slot with the address of the current implementation.
27      * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1,
28      * and is
29      * validated in the constructor.
30      */
31     bytes32 internal constant _IMPLEMENTATION_SLOT =
32         0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc;
33
34     /**
35      * @dev Emitted when the implementation is upgraded.
36      */
37     event Upgraded(address indexed implementation);
38
39     /**
40      * @dev Returns the current implementation address.
41      */
42     function _getImplementation() internal view returns (address) {
43         return StorageSlot.getAddressSlot(_IMPLEMENTATION_SLOT).value;
44     }
45
46     /**
47      * @dev Stores a new address in the EIP1967 implementation slot.
48      */
49     function _setImplementation(address newImplementation) private {
50         require(Address.isContract(newImplementation), "ERC1967: new implementation
51             is not a contract");
52         StorageSlot.getAddressSlot(_IMPLEMENTATION_SLOT).value = newImplementation;
53     }
54
55     /**
56      * @dev Perform implementation upgrade
57      *
58      * Emits an {Upgraded} event.
59      */
60     function _upgradeTo(address newImplementation) internal {
61         _setImplementation(newImplementation);
62         emit Upgraded(newImplementation);
63     }
64
65     /**
66      * @dev Perform implementation upgrade with additional setup call.
67      *
68      * Emits an {Upgraded} event.
69      */
70     function _upgradeToAndCall(
71         address newImplementation,
72         bytes memory data,
73         bool forceCall
74     )

```

```

69     ) internal {
70         _upgradeTo(newImplementation);
71         if (data.length > 0 || forceCall) {
72             Address.functionDelegateCall(newImplementation, data);
73         }
74     }
75
76 /**
77  * @dev Perform implementation upgrade with security checks for UUPS proxies, and
78  * additional setup call.
79  * Emits an {Upgraded} event.
80  */
81 function _upgradeToAndCallUUPS(
82     address newImplementation,
83     bytes memory data,
84     bool forceCall
85 ) internal {
86     // Upgrades from old implementations will perform a rollback test. This test
87     // requires the new
88     // implementation to upgrade back to the old, non-ERC1822 compliant,
89     // implementation. Removing
90     // this special case will break upgrade paths from old UUPS implementation to
91     // new ones.
92     if (StorageSlot.getBooleanSlot(_ROLLBACK_SLOT).value) {
93         _setImplementation(newImplementation);
94     } else {
95         try IERC1822Proxiable(newImplementation).proxiableUUID() returns (bytes32
96             slot) {
97             require(slot == _IMPLEMENTATION_SLOT, "ERC1967Upgrade: unsupported
98                 proxiableUUID");
99         } catch {
100             revert("ERC1967Upgrade: new implementation is not UUPS");
101         }
102         _upgradeToAndCall(newImplementation, data, forceCall);
103     }
104 }
105
106 /**
107  * @dev Storage slot with the admin of the contract.
108  * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
109  * validated in the constructor.
110  */
111 bytes32 internal constant _ADMIN_SLOT =
112     0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;
113
114 /**
115  * @dev Emitted when the admin account has changed.
116  */
117 event AdminChanged(address previousAdmin, address newAdmin);
118
119 /**
120  * @dev Returns the current admin.
121  */
122 function _getAdmin() internal view returns (address) {
123     return StorageSlot.getAddressSlot(_ADMIN_SLOT).value;
124 }
125
126 /**
127  * @dev Stores a new address in the EIP1967 admin slot.
128  */
129 function _setAdmin(address newAdmin) private {
130     require(newAdmin != address(0), "ERC1967: new admin is the zero address");
131     StorageSlot.getAddressSlot(_ADMIN_SLOT).value = newAdmin;
132 }
133
134 /**
135  * @dev Changes the admin of the proxy.
136  * Emits an {AdminChanged} event.
137  */
138 function _changeAdmin(address newAdmin) internal {
139     emit AdminChanged(_getAdmin(), newAdmin);

```

```

135     _setAdmin(newAdmin);
136 }
137
138 /**
139  * @dev The storage slot of the UpgradeableBeacon contract which defines the
140  * implementation for this proxy.
141  * This is bytes32(uint256(keccak256('eip1967.proxy.beacon')) - 1)) and is
142  * validated in the constructor.
143  */
144 bytes32 internal constant _BEACON_SLOT =
145     0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaae59ff6cb3582b35133d50;
146
147 /**
148  * @dev Emitted when the beacon is upgraded.
149  */
150 event BeaconUpgraded(address indexed beacon);
151
152 /**
153  * @dev Returns the current beacon.
154  */
155 function _getBeacon() internal view returns (address) {
156     return StorageSlot.getAddressSlot(_BEACON_SLOT).value;
157 }
158
159 /**
160  * @dev Stores a new beacon in the EIP1967 beacon slot.
161  */
162 function _setBeacon(address newBeacon) private {
163     require(Address.isContract(newBeacon), "ERC1967: new beacon is not a contract");
164     require(
165         Address.isContract(IBeacon(newBeacon).implementation()),
166         "ERC1967: beacon implementation is not a contract"
167     );
168     StorageSlot.getAddressSlot(_BEACON_SLOT).value = newBeacon;
169 }
170
171 /**
172  * @dev Perform beacon upgrade with additional setup call. Note: This upgrades
173  * the address of the beacon, it does
174  * not upgrade the implementation contained in the beacon (see
175  * {UpgradeableBeacon-_setImplementation} for that).
176  *
177  * @dev Emits a {BeaconUpgraded} event.
178  */
179 function _upgradeBeaconToAndCall(
180     address newBeacon,
181     bytes memory data,
182     bool forceCall
183 ) internal {
184     _setBeacon(newBeacon);
185     emit BeaconUpgraded(newBeacon);
186     if (data.length > 0 || forceCall) {
187         Address.functionDelegateCall(IBeacon(newBeacon).implementation(), data);
188     }
189 }

```