Full Verified Source Code - Clanker LP Locker v2

The permanent liquidity locker contract that holds $DRB liquidity

Locker Contract Address: 0x5ec4f99f342038c67a312a166ff56e6d70383d86
Verified on Basescan:
https://basescan.org/address/0x5ec4f99f342038c67a312a166ff56e6d70383d86#code
$DRB Token Contract (for reference): 0x3ec2156d4c0a9cbdab4a016633b7bcf6a8d68ea2

Contract Source Code (Solidity Standard Json-Input format)
File 1 of 10 : LpLockerv2.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

import "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Context.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import {IERC721Receiver} from
"@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
import {NonFungibleContract} from "./IManager.sol";

contract LpLockerv2 is Ownable, IERC721Receiver {
    event LockId(uint256 _id);
    event Received(address indexed from, uint256 tokenId);

    error NotAllowed(address user);
    error InvalidTokenId(uint256 tokenId);

    event ClaimedRewards(
        address indexed claimer,
        address indexed token0,
        address indexed token1,
        uint256 amount0,
        uint256 amount1,
        uint256 totalAmount1,
        uint256 totalAmount0
    );

    IERC721 private SafeERC721;
```

```
address private immutable e721Token;
address public positionManager = 0x03a520b32C04BF3bEEf7BEb72E919cf822Ed34f1;
string public constant version = "0.0.2";
uint256 public _clankerTeamReward;
address public _clankerTeamRecipient;
address public _factory;
struct UserRewardRecipient {
    address recipient;
    uint256 lpTokenId;
}

struct TeamRewardRecipient {
    address recipient;
    uint256 reward;
    uint256 lpTokenId;
}

mapping(uint256 => UserRewardRecipient) public _userRewardRecipientForToken;
mapping(uint256 => TeamRewardRecipient)
    public _teamOverrideRewardRecipientForToken;

mapping(address => uint256[]) public _userTokenIds;

constructor(
    address tokenFactory, // Address of the clanker factory
    address token, // Address of the ERC721 Uniswap V3 LP NFT
    address clankerTeamRecipient, // clanker team address to receive portion of the fees
    uint256 clankerTeamReward // clanker team reward percentage
) Ownable(clankerTeamRecipient) {
    SafeERC721 = IERC721(token);
    e721Token = token;
    _factory = tokenFactory;
    _clankerTeamReward = clankerTeamReward;
    _clankerTeamRecipient = clankerTeamRecipient;
}

modifier onlyOwnerOrFactory() {
    if (msg.sender != owner() && msg.sender != _factory) {
        revert NotAllowed(msg.sender);
    }
    _;
}

function setOverrideTeamRewardsForToken(
```

```
    uint256 tokenId,
    address newTeamRecipient,
    uint256 newTeamReward
) public onlyOwner {
    _teamOverrideRewardRecipientForToken[tokenId] = TeamRewardRecipient({
        recipient: newTeamRecipient,
        reward: newTeamReward,
        lpTokenId: tokenId
    });
}

function updateClankerFactory(address newFactory) public onlyOwner {
    _factory = newFactory;
}

// Update the clanker team reward
function updateClankerTeamReward(uint256 newReward) public onlyOwner {
    _clankerTeamReward = newReward;
}

// Update the clanker team recipient
function updateClankerTeamRecipient(address newRecipient) public onlyOwner {
    _clankerTeamRecipient = newRecipient;
}

// Withdraw ETH from the contract
function withdrawETH(address recipient) public onlyOwner {
    payable(recipient).transfer(address(this).balance);
}

// Withdraw ERC20 tokens from the contract
function withdrawERC20(address _token, address recipient) public onlyOwner {
    IERC20 IToken = IERC20(_token);
    IToken.transfer(recipient, IToken.balanceOf(address(this)));
}

// Use collect rewards to collect the rewards
function collectRewards(uint256 _tokenId) public {
    // Get the _userRewardRecipients for the tokenId
    UserRewardRecipient
        memory userRewardRecipient = _userRewardRecipientForToken[_tokenId];

    address _recipient = userRewardRecipient.recipient;
```

```
if (_recipient == address(0)) {
    revert InvalidTokenId(_tokenId);
}

NonFungibleContract nonfungiblePositionManager = NonFungibleContract(
    positionManager
);

(uint256 amount0, uint256 amount1) = nonfungiblePositionManager.collect(
    NonFungibleContract.CollectParams({
        recipient: address(this),
        amount0Max: type(uint128).max,
        amount1Max: type(uint128).max,
        tokenId: _tokenId
    })
);

(
    ,
    ,
    address token0,
    address token1,
    ,
    ,
    ,
    ,
    ,
    ,
    ,
    ,

) = nonfungiblePositionManager.positions(_tokenId);

IERC20 rewardToken0 = IERC20(token0);
IERC20 rewardToken1 = IERC20(token1);

address teamRecipient = _clankerTeamRecipient;
uint256 teamReward = _clankerTeamReward;

TeamRewardRecipient
    memory overrideRewardRecipient = _teamOverrideRewardRecipientForToken[
        _tokenId
    ];

if (overrideRewardRecipient.recipient != address(0)) {
```

```
            teamRecipient = overrideRewardRecipient.recipient;
            teamReward = overrideRewardRecipient.reward;
        }

        uint256 protocolReward0 = (amount0 * teamReward) / 100;
        uint256 protocolReward1 = (amount1 * teamReward) / 100;

        uint256 recipientReward0 = amount0 - protocolReward0;
        uint256 recipientReward1 = amount1 - protocolReward1;

        rewardToken0.transfer(_recipient, recipientReward0);
        rewardToken1.transfer(_recipient, recipientReward1);

        rewardToken0.transfer(teamRecipient, protocolReward0);
        rewardToken1.transfer(teamRecipient, protocolReward1);

        emit ClaimedRewards(
            _recipient,
            token0,
            token1,
            recipientReward0,
            recipientReward1,
            amount0,
            amount1
        );
    }

    function getLpTokenIdsForUser(
        address user
    ) public view returns (uint256[] memory) {
        return _userTokenIds[user];
    }

    function addUserRewardRecipient(
        UserRewardRecipient memory recipient
    ) public onlyOwnerOrFactory {
        _userRewardRecipientForToken[recipient.lpTokenId] = recipient;
        _userTokenIds[recipient.recipient].push(recipient.lpTokenId);
    }

    function replaceUserRewardRecipient(
        UserRewardRecipient memory recipient
    ) public {
        // Get the old recipient
```

```
        UserRewardRecipient memory oldRecipient = _userRewardRecipientForToken[
            recipient.lpTokenId
        ];

        // Only owner or recipient can replace the reward recipient
        if (msg.sender != owner() && msg.sender != oldRecipient.recipient) {
            revert NotAllowed(msg.sender);
        }

        // Remove the old recipient
        delete _userRewardRecipientForToken[recipient.lpTokenId];

        // Remove the old tokenId from _userTokenIds
        uint256[] memory tokenIds = _userTokenIds[recipient.recipient];
        for (uint256 i = 0; i < tokenIds.length; i++) {
            if (tokenIds[i] == recipient.lpTokenId) {
                delete _userTokenIds[recipient.recipient][i];
            }
        }

        // Add the new recipient
        _userRewardRecipientForToken[recipient.lpTokenId] = recipient;

        // Add the new tokenId to _userTokenIds
        _userTokenIds[recipient.recipient].push(recipient.lpTokenId);
    }

    function onERC721Received(
        address,
        address from,
        uint256 id,
        bytes calldata
    ) external override returns (bytes4) {
        // Only clanker team EOA can send the NFT here
        if (from != _factory) {
            revert NotAllowed(from);
        }

        emit Received(from, id);
        return IERC721Receiver.onERC721Received.selector;
    }
}
```

File 2 of 10 : IERC721.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v5.1.0) (token/ERC721/IERC721.sol)

pragma solidity ^0.8.20;

import {IERC165} from "../../utils/introspection/IERC165.sol";

/**
 * @dev Required interface of an ERC-721 compliant contract.
 */
interface IERC721 is IERC165 {
    /**
     * @dev Emitted when `tokenId` token is transferred from `from` to `to`.
     */
    event Transfer(address indexed from, address indexed to, uint256 indexed tokenId);

    /**
     * @dev Emitted when `owner` enables `approved` to manage the `tokenId` token.
     */
    event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);

    /**
     * @dev Emitted when `owner` enables or disables (`approved`) `operator` to manage all of its
assets.
     */
    event ApprovalForAll(address indexed owner, address indexed operator, bool approved);

    /**
     * @dev Returns the number of tokens in ``owner``'s account.
     */
    function balanceOf(address owner) external view returns (uint256 balance);

    /**
     * @dev Returns the owner of the `tokenId` token.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
```

```
    */
    function ownerOf(uint256 tokenId) external view returns (address owner);

    /**
     * @dev Safely transfers `tokenId` token from `from` to `to`.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by `from`.
     * - If the caller is not `from`, it must be approved to move this token by either {approve} or
{setApprovalForAll}.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received},
which is called upon
     *   a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(address from, address to, uint256 tokenId, bytes calldata data)
external;

    /**
     * @dev Safely transfers `tokenId` token from `from` to `to`, checking first that contract
recipients
     * are aware of the ERC-721 protocol to prevent tokens from being forever locked.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by `from`.
     * - If the caller is not `from`, it must have been allowed to move this token by either {approve}
or
     *   {setApprovalForAll}.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received},
which is called upon
     *   a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(address from, address to, uint256 tokenId) external;
```

```
    /**
     * @dev Transfers `tokenId` token from `from` to `to`.
     *
     * WARNING: Note that the caller is responsible to confirm that the recipient is capable of receiving ERC-721
     * or else they may be permanently lost. Usage of {safeTransferFrom} prevents loss, though the caller must
     * understand this adds an external call which potentially creates a reentrancy vulnerability.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must be owned by `from`.
     * - If the caller is not `from`, it must be approved to move this token by either {approve} or {setApprovalForAll}.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address from, address to, uint256 tokenId) external;

    /**
     * @dev Gives permission to `to` to transfer `tokenId` token to another account.
     * The approval is cleared when the token is transferred.
     *
     * Only a single account can be approved at a time, so approving the zero address clears previous approvals.
     *
     * Requirements:
     *
     * - The caller must own the token or be an approved operator.
     * - `tokenId` must exist.
     *
     * Emits an {Approval} event.
     */
    function approve(address to, uint256 tokenId) external;

    /**
     * @dev Approve or remove `operator` as an operator for the caller.
     * Operators can call {transferFrom} or {safeTransferFrom} for any token owned by the caller.
     *
```

```
     * Requirements:
     *
     * - The `operator` cannot be the address zero.
     *
     * Emits an {ApprovalForAll} event.
     */
    function setApprovalForAll(address operator, bool approved) external;

    /**
     * @dev Returns the account approved for `tokenId` token.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     */
    function getApproved(uint256 tokenId) external view returns (address operator);

    /**
     * @dev Returns if the `operator` is allowed to manage all of the assets of `owner`.
     *
     * See {setApprovalForAll}
     */
    function isApprovedForAll(address owner, address operator) external view returns (bool);
}
```

File 3 of 10 : IERC20.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v5.1.0) (token/ERC20/IERC20.sol)

pragma solidity ^0.8.20;

/**
 * @dev Interface of the ERC-20 standard as defined in the ERC.
 */
interface IERC20 {
    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
```

```
 */
event Transfer(address indexed from, address indexed to, uint256 value);

/**
 * @dev Emitted when the allowance of a `spender` for an `owner` is set by
 * a call to {approve}. `value` is the new allowance.
 */
event Approval(address indexed owner, address indexed spender, uint256 value);

/**
 * @dev Returns the value of tokens in existence.
 */
function totalSupply() external view returns (uint256);

/**
 * @dev Returns the value of tokens owned by `account`.
 */
function balanceOf(address account) external view returns (uint256);

/**
 * @dev Moves a `value` amount of tokens from the caller's account to `to`.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
function transfer(address to, uint256 value) external returns (bool);

/**
 * @dev Returns the remaining number of tokens that `spender` will be
 * allowed to spend on behalf of `owner` through {transferFrom}. This is
 * zero by default.
 *
 * This value changes when {approve} or {transferFrom} are called.
 */
function allowance(address owner, address spender) external view returns (uint256);

/**
 * @dev Sets a `value` amount of tokens as the allowance of `spender` over the
 * caller's tokens.
 *
 * Returns a boolean value indicating whether the operation succeeded.
```

```
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 value) external returns (bool);

    /**
     * @dev Moves a `value` amount of tokens from `from` to `to` using the
     * allowance mechanism. `value` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address from, address to, uint256 value) external returns (bool);
}
```

File 4 of 10 : Ownable.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v5.0.0) (access/Ownable.sol)

pragma solidity ^0.8.20;

import {Context} from "../utils/Context.sol";

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * The initial owner is set to the address provided by the deployer. This can
 * later be changed with {transferOwnership}.
 *
```

```
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

    /**
     * @dev The caller account is not authorized to perform an operation.
     */
    error OwnableUnauthorizedAccount(address account);

    /**
     * @dev The owner is not a valid owner account. (eg. `address(0)`)
     */
    error OwnableInvalidOwner(address owner);

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the address provided by the deployer as the initial
owner.
     */
    constructor(address initialOwner) {
        if (initialOwner == address(0)) {
            revert OwnableInvalidOwner(address(0));
        }
        _transferOwnership(initialOwner);
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        _checkOwner();
        _;
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
```

```
    return _owner;
}

/**
 * @dev Throws if the sender is not the owner.
 */
function _checkOwner() internal view virtual {
    if (owner() != _msgSender()) {
        revert OwnableUnauthorizedAccount(_msgSender());
    }
}

/**
 * @dev Leaves the contract without owner. It will not be possible to call
 * `onlyOwner` functions. Can only be called by the current owner.
 *
 * NOTE: Renouncing ownership will leave the contract without an owner,
 * thereby disabling any functionality that is only available to the owner.
 */
function renounceOwnership() public virtual onlyOwner {
    _transferOwnership(address(0));
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address newOwner) public virtual onlyOwner {
    if (newOwner == address(0)) {
        revert OwnableInvalidOwner(address(0));
    }
    _transferOwnership(newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Internal function without access restriction.
 */
function _transferOwnership(address newOwner) internal virtual {
    address oldOwner = _owner;
    _owner = newOwner;
    emit OwnershipTransferred(oldOwner, newOwner);
```

```
    }
}
```

File 5 of 10 : Context.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v5.0.1) (utils/Context.sol)

pragma solidity ^0.8.20;

/**
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }

    function _contextSuffixLength() internal view virtual returns (uint256) {
        return 0;
    }
}
```

File 6 of 10 : Address.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v5.1.0) (utils/Address.sol)

pragma solidity ^0.8.20;
```

```
import {Errors} from "./Errors.sol";

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev There's no code at `target` (it is not a contract).
     */
    error AddressEmptyCode(address target);

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://consensys.net/diligence/blog/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.8.20/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        if (address(this).balance < amount) {
            revert Errors.InsufficientBalance(address(this).balance, amount);
        }

        (bool success, ) = recipient.call{value: amount}("");
        if (!success) {
            revert Errors.FailedCall();
        }
    }
```

```
    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain `call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason or custom error, it is bubbled
     * up by this function (like regular Solidity function calls). However, if
     * the call reverted with no returned reason, this function reverts with a
     * {Errors.FailedCall} error.
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use
https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#
abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal
returns (bytes memory) {
        if (address(this).balance < value) {
            revert Errors.InsufficientBalance(address(this).balance, value);
        }
        (bool success, bytes memory returndata) = target.call{value: value}(data);
        return verifyCallResultFromTarget(target, success, returndata);
    }
```

```
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes
memory) {
        (bool success, bytes memory returndata) = target.staticcall(data);
        return verifyCallResultFromTarget(target, success, returndata);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes
memory) {
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return verifyCallResultFromTarget(target, success, returndata);
    }

    /**
     * @dev Tool to verify that a low level call to smart-contract was successful, and reverts if the
target
     * was not a contract or bubbling up the revert reason (falling back to {Errors.FailedCall}) in
case
     * of an unsuccessful call.
     */
    function verifyCallResultFromTarget(
        address target,
        bool success,
        bytes memory returndata
    ) internal view returns (bytes memory) {
        if (!success) {
            _revert(returndata);
        } else {
            // only check if target is a contract if the call was successful and the return data is empty
            // otherwise we already know that it was a contract
            if (returndata.length == 0 && target.code.length == 0) {
                revert AddressEmptyCode(target);
            }
            return returndata;
        }
```

```
    }

    /**
     * @dev Tool to verify that a low level call was successful, and reverts if it wasn't, either by bubbling the
     * revert reason or with a default {Errors.FailedCall} error.
     */
    function verifyCallResult(bool success, bytes memory returndata) internal pure returns (bytes memory) {
        if (!success) {
            _revert(returndata);
        } else {
            return returndata;
        }
    }

    /**
     * @dev Reverts with returndata if present. Otherwise reverts with {Errors.FailedCall}.
     */
    function _revert(bytes memory returndata) private pure {
        // Look for revert reason and bubble it up if present
        if (returndata.length > 0) {
            // The easiest way to bubble the revert reason is using memory via assembly
            assembly ("memory-safe") {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
        } else {
            revert Errors.FailedCall();
        }
    }
}
```

File 7 of 10 : IERC721Receiver.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v5.1.0) (token/ERC721/IERC721Receiver.sol)

pragma solidity ^0.8.20;

/**
```

```
 * @title ERC-721 token receiver interface
 * @dev Interface for any contract that wants to support safeTransfers
 * from ERC-721 asset contracts.
 */
interface IERC721Receiver {
    /**
     * @dev Whenever an {IERC721} `tokenId` token is transferred to this contract via
{IERC721-safeTransferFrom}
     * by `operator` from `from`, this function is called.
     *
     * It must return its Solidity selector to confirm the token transfer.
     * If any other value is returned or the interface is not implemented by the recipient, the
transfer will be
     * reverted.
     *
     * The selector can be obtained in Solidity with
`IERC721Receiver.onERC721Received.selector`.
     */
    function onERC721Received(
        address operator,
        address from,
        uint256 tokenId,
        bytes calldata data
    ) external returns (bytes4);
}
```

File 8 of 10 : IManager.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.25;

import {IERC721} from "@openzeppelin/contracts/token/ERC721/IERC721.sol";

interface NonFungibleContract is IERC721 {
    /// @notice Returns the position information associated with a given token ID.
    /// @dev Throws if the token ID is not valid.
    /// @param tokenId The ID of the token that represents the position
    /// @return nonce The nonce for permits
    /// @return operator The address that is approved for spending
    /// @return token0 The address of the token0 for a specific pool
```

```
    /// @return token1 The address of the token1 for a specific pool
    /// @return fee The fee associated with the pool
    /// @return tickLower The lower end of the tick range for the position
    /// @return tickUpper The higher end of the tick range for the position
    /// @return liquidity The liquidity of the position
    /// @return feeGrowthInside0LastX128 The fee growth of token0 as of the last action on the
individual position
    /// @return feeGrowthInside1LastX128 The fee growth of token1 as of the last action on the
individual position
    /// @return tokensOwed0 The uncollected amount of token0 owed to the position as of the
last computation
    /// @return tokensOwed1 The uncollected amount of token1 owed to the position as of the
last computation
    function positions(
        uint256 tokenId
    )
        external
        view
        returns (
            uint96 nonce,
            address operator,
            address token0,
            address token1,
            uint24 fee,
            int24 tickLower,
            int24 tickUpper,
            uint128 liquidity,
            uint256 feeGrowthInside0LastX128,
            uint256 feeGrowthInside1LastX128,
            uint128 tokensOwed0,
            uint128 tokensOwed1
        );

    struct CollectParams {
        uint256 tokenId;
        address recipient;
        uint128 amount0Max;
        uint128 amount1Max;
    }

    /// @notice Collects up to a maximum amount of fees owed to a specific position to the
recipient
```

```
    /// @param params tokenId The ID of the NFT for which tokens are being collected,
    /// recipient The account that should receive the tokens,
    /// amount0Max The maximum amount of token0 to collect,
    /// amount1Max The maximum amount of token1 to collect
    /// @return amount0 The amount of fees collected in token0
    /// @return amount1 The amount of fees collected in token1
    function collect(
        CollectParams calldata params
    ) external payable returns (uint256 amount0, uint256 amount1);
}
```

File 9 of 10 : IERC165.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v5.1.0) (utils/introspection/IERC165.sol)

pragma solidity ^0.8.20;

/**
 * @dev Interface of the ERC-165 standard, as defined in the
 * https://eips.ethereum.org/EIPS/eip-165[ERC].
 *
 * Implementers can declare support of contract interfaces, which can then be
 * queried by others ({ERC165Checker}).
 *
 * For an implementation, see {ERC165}.
 */
interface IERC165 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * https://eips.ethereum.org/EIPS/eip-165#how-interfaces-are-identified[ERC section]
     * to learn more about how these ids are created.
     *
     * This function call must use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}
```

File 10 of 10 : Errors.sol

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v5.1.0) (utils/Errors.sol)

pragma solidity ^0.8.20;

/**
 * @dev Collection of common custom errors used in multiple contracts
 *
 * IMPORTANT: Backwards compatibility is not guaranteed in future versions of the library.
 * It is recommended to avoid relying on the error API for critical functionality.
 *
 * _Available since v5.1._
 */
library Errors {
    /**
     * @dev The ETH balance of the account is not enough to perform the operation.
     */
    error InsufficientBalance(uint256 balance, uint256 needed);

    /**
     * @dev A call to an address target failed. The target may have reverted.
     */
    error FailedCall();

    /**
     * @dev The deployment failed.
     */
    error FailedDeployment();

    /**
     * @dev A necessary precompile is missing.
     */
    error MissingPrecompile(address);
}
```

Settings

```
{
  "remappings": [
    "@openzeppelin/contracts/=lib/openzeppelin-contracts/contracts/",
    "@uniswap/v3-core/=lib/v3-core/",
    "@solady/=lib/optimism/packages/contracts-bedrock/lib/solady/src/",
```

```
    "@solady-v0.0.245/=lib/optimism/packages/contracts-bedrock/lib/solady/src/",
    "@contracts-bedrock/=lib/optimism/packages/contracts-bedrock/",
    "automate/=lib/optimism/packages/contracts-bedrock/lib/automate/contracts/",
    "ds-test/=lib/openzeppelin-contracts/lib/forge-std/lib/ds-test/src/",
    "erc4626-tests/=lib/openzeppelin-contracts/lib/erc4626-tests/",
    "forge-std/=lib/forge-std/src/",
    "halmos-cheatcodes/=lib/openzeppelin-contracts/lib/halmos-cheatcodes/src/",

"kontrol-cheatcodes/=lib/optimism/packages/contracts-bedrock/lib/kontrol-cheatcodes/src/",
    "lib-keccak/=lib/optimism/packages/contracts-bedrock/lib/lib-keccak/contracts/",

"openzeppelin-contracts-upgradeable/=lib/optimism/packages/contracts-bedrock/lib/openzeppelin-contracts-upgradeable/",

"openzeppelin-contracts-v5/=lib/optimism/packages/contracts-bedrock/lib/openzeppelin-contracts-v5/",
    "openzeppelin-contracts/=lib/openzeppelin-contracts/",
    "optimism/=lib/optimism/",
    "prb-test/=lib/optimism/packages/contracts-bedrock/lib/automate/lib/prb-test/src/",
    "safe-contracts/=lib/optimism/packages/contracts-bedrock/lib/safe-contracts/contracts/",
    "solady-v0.0.245/=lib/optimism/packages/contracts-bedrock/lib/solady-v0.0.245/src/",
    "solady/=lib/solady/src/",
    "solmate/=lib/optimism/packages/contracts-bedrock/lib/solmate/src/",
    "v3-core/=lib/v3-core/"
  ],
  "optimizer": {
    "enabled": true,
    "runs": 200
  },
  "metadata": {
    "useLiteralContent": false,
    "bytecodeHash": "ipfs",
    "appendCBOR": true
  },
  "outputSelection": {
    "*": {
      "*": [
        "evm.bytecode",
        "evm.deployedBytecode",
        "devdoc",
        "userdoc",
        "metadata",
```

```
        "abi"
      ]
    }
  },
  "evmVersion": "cancun",
  "viaIR": true,
  "libraries": {}
}
```