

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3
4 library LowGasSafeMath {
5     /// @notice Returns x + y, reverts if sum overflows
6     uint256
7     /// @param x The augend
8     /// @param y The addend
9     /// @return z The sum of x and y
10    function add(uint256 x, uint256 y) internal pure
11    returns (uint256 z) {
12        require((z = x + y) >= x);
13    }
14
15    function add32(uint32 x, uint32 y) internal pure
16    returns (uint32 z) {
17        require((z = x + y) >= x);
18    }
19
20    /// @notice Returns x - y, reverts if underflows
21    uint256
22    /// @param x The minuend
23    /// @param y The subtrahend
24    /// @return z The difference of x and y
25    function sub(uint256 x, uint256 y) internal pure
26    returns (uint256 z) {
27        require((z = x - y) <= x);
28    }
29
30    function sub32(uint32 x, uint32 y) internal pure
31    returns (uint32 z) {
32        require((z = x - y) <= x);
33    }
34
35    /// @notice Returns x * y, reverts if overflows
36    /// @param x The multiplicand
37    /// @param y The multiplier
38    /// @return z The product of x and y
39    function mul(uint256 x, uint256 y) internal pure
40    returns (uint256 z) {
41        require(x == 0 || (z = x * y) / x == y);
42    }
43
44    /// @notice Returns x + y, reverts if overflows
45    or underflows
46    /// @param x The augend
47    /// @param y The addend
48    /// @return z The sum of x and y
49    function add(int256 x, int256 y) internal pure
50    returns (int256 z) {
51        require((z = x + y) >= x == (y >= 0));
52    }
53
54    /// @notice Returns x - y, reverts if overflows
55    or underflows
56    /// @param x The minuend
57    /// @param y The subtrahend
58    /// @return z The difference of x and y
59    function sub(int256 x, int256 y) internal pure
60    returns (int256 z) {
61        require((z = x - y) <= x == (y >= 0));
62    }
63 }

```

```

1 // SPDX-License-Identifier: AGPL-3.0-or-later
2 pragma solidity 0.7.5;
3 // Only change to generate diff
4 library LowGasSafeMath {
5     /// @notice Returns x + y, reverts if sum overflows
6     uint256
7     /// @param x The augend
8     /// @param y The addend
9     /// @return z The sum of x and y
10    function add(uint256 x, uint256 y) internal pure
11    returns (uint256 z) {
12        require((z = x + y) >= x);
13    }
14
15    function add32(uint32 x, uint32 y) internal pure
16    returns (uint32 z) {
17        require((z = x + y) >= x);
18    }
19
20    /// @notice Returns x - y, reverts if underflows
21    uint256
22    /// @param x The minuend
23    /// @param y The subtrahend
24    /// @return z The difference of x and y
25    function sub(uint256 x, uint256 y) internal pure
26    returns (uint256 z) {
27        require((z = x - y) <= x);
28    }
29
30    function sub32(uint32 x, uint32 y) internal pure
31    returns (uint32 z) {
32        require((z = x - y) <= x);
33    }
34
35    /// @notice Returns x * y, reverts if overflows
36    /// @param x The multiplicand
37    /// @param y The multiplier
38    /// @return z The product of x and y
39    function mul(uint256 x, uint256 y) internal pure
40    returns (uint256 z) {
41        require(x == 0 || (z = x * y) / x == y);
42    }
43
44    /// @notice Returns x + y, reverts if overflows
45    or underflows
46    /// @param x The augend
47    /// @param y The addend
48    /// @return z The sum of x and y
49    function add(int256 x, int256 y) internal pure
50    returns (int256 z) {
51        require((z = x + y) >= x == (y >= 0));
52    }
53
54    /// @notice Returns x - y, reverts if overflows
55    or underflows
56    /// @param x The minuend
57    /// @param y The subtrahend
58    /// @return z The difference of x and y
59    function sub(int256 x, int256 y) internal pure
60    returns (int256 z) {
61        require((z = x - y) <= x == (y >= 0));
62    }
63 }

```

```

52
53 interface IERC20 {
54     function decimals() external view returns (uint
55         8);
56     /**
57      * @dev Returns the amount of tokens in existenc
58      e.
59      */
60     function totalSupply() external view returns (uin
61         t256);
62     /**
63      * @dev Returns the amount of tokens owned by `ac
64      count`.
65      */
66     function balanceOf(address account) external view
67         returns (uint256);
68     /**
69      * @dev Moves `amount` tokens from the caller's a
70      ccount to `recipient`.
71      *
72      * Returns a boolean value indicating whether the
73      operation succeeded.
74      *
75      * Emits a {Transfer} event.
76      */
77     function transfer(address recipient, uint256 amou
78         nt) external returns (bool);
79     /**
80      * @dev Returns the remaining number of tokens th
81      at `spender` will be
82      * allowed to spend on behalf of `owner` through
83      {transferFrom}. This is
84      * zero by default.
85      *
86      * This value changes when {approve} or {transfer
87      From} are called.
88      */
89     function allowance(address owner, address spende
90         r) external view returns (uint256);
91     /**
92      * @dev Sets `amount` as the allowance of `spende
93      r` over the caller's tokens.
94      *
95      * Returns a boolean value indicating whether the
96      operation succeeded.
97      *
98      * IMPORTANT: Beware that changing an allowance w
99      ith this method brings the risk
100     * that someone may use both the old and the new
101     allowance by unfortunate
102     transaction ordering. One possible solution to
103     mitigate this race
104     condition is to first reduce the spender's all
105     owance to 0 and set the
106     * desired value afterwards:
107     * https://github.com/ethereum/EIPs/issues/20#iss
108     uecomment-263524729
109     */
110     function approve(address spender, uint256 amount)
111         external returns (bool);
112     /**

```

```

53
54 interface IERC20 {
55     function decimals() external view returns (uint
56         8);
57     /**
58      * @dev Returns the amount of tokens in existenc
59      e.
60      */
61     function totalSupply() external view returns (uin
62         t256);
63     /**
64      * @dev Returns the amount of tokens owned by `ac
65      count`.
66      */
67     function balanceOf(address account) external view
68         returns (uint256);
69     /**
70      * @dev Moves `amount` tokens from the caller's a
71      ccount to `recipient`.
72      *
73      * Returns a boolean value indicating whether the
74      operation succeeded.
75      *
76      * Emits a {Transfer} event.
77      */
78     function transfer(address recipient, uint256 amou
79         nt) external returns (bool);
80     /**
81      * @dev Returns the remaining number of tokens th
82      at `spender` will be
83      * allowed to spend on behalf of `owner` through
84      {transferFrom}. This is
85      * zero by default.
86      *
87      * This value changes when {approve} or {transfer
88      From} are called.
89      */
90     function allowance(address owner, address spende
91         r) external view returns (uint256);
92     /**
93      * @dev Sets `amount` as the allowance of `spende
94      r` over the caller's tokens.
95      *
96      * Returns a boolean value indicating whether the
97      operation succeeded.
98      *
99      * IMPORTANT: Beware that changing an allowance w
100     ith this method brings the risk
101     * that someone may use both the old and the new
102     allowance by unfortunate
103     transaction ordering. One possible solution to
104     mitigate this race
105     condition is to first reduce the spender's all
106     owance to 0 and set the
107     * desired value afterwards:
108     * https://github.com/ethereum/EIPs/issues/20#iss
109     uecomment-263524729
110     */
111     function approve(address spender, uint256 amount)
112         external returns (bool);
113     /**

```

```

100 * @dev Moves `amount` tokens from `sender` to `recipient` using the
101 * allowance mechanism. `amount` is then deducted from the caller's
102 * allowance.
103 *
104 * Returns a boolean value indicating whether the operation succeeded.
105 *
106 * Emits a {Transfer} event.
107 */
108 function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
109
110 /**
111 * @dev Emitted when `value` tokens are moved from one account (`from`) to
112 * another (`to`).
113 *
114 * Note that `value` may be zero.
115 */
116 event Transfer(address indexed from, address indexed to, uint256 value);
117
118 /**
119 * @dev Emitted when the allowance of a `spender` for an `owner` is set by
120 * a call to {approve}. `value` is the new allowance.
121 */
122 event Approval(address indexed owner, address indexed spender, uint256 value);
123 }
124
125 library Address {
126     /**
127     * @dev Returns true if `account` is a contract.
128     *
129     * [IMPORTANT]
130     * ====
131     * It is unsafe to assume that an address for which this function returns
132     * false is an externally-owned account (EOA) and not a contract.
133     *
134     * Among others, `isContract` will return false for the following
135     * types of addresses:
136     *
137     * - an externally-owned account
138     * - a contract in construction
139     * - an address where a contract will be created
140     * - an address where a contract lived, but was destroyed
141     *
142     * ====
143     */
144     function isContract(address account) internal view returns (bool) {
145         // This method relies in extcodesize, which returns 0 for contracts in
146         // construction, since the code is only stored at the end of the
147         // constructor execution.
148         uint256 size;

```

```

101 * @dev Moves `amount` tokens from `sender` to `recipient` using the
102 * allowance mechanism. `amount` is then deducted from the caller's
103 * allowance.
104 *
105 * Returns a boolean value indicating whether the operation succeeded.
106 *
107 * Emits a {Transfer} event.
108 */
109 function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
110
111 /**
112 * @dev Emitted when `value` tokens are moved from one account (`from`) to
113 * another (`to`).
114 *
115 * Note that `value` may be zero.
116 */
117 event Transfer(address indexed from, address indexed to, uint256 value);
118
119 /**
120 * @dev Emitted when the allowance of a `spender` for an `owner` is set by
121 * a call to {approve}. `value` is the new allowance.
122 */
123 event Approval(address indexed owner, address indexed spender, uint256 value);
124 }
125
126 library Address {
127     /**
128     * @dev Returns true if `account` is a contract.
129     *
130     * [IMPORTANT]
131     * ====
132     * It is unsafe to assume that an address for which this function returns
133     * false is an externally-owned account (EOA) and not a contract.
134     *
135     * Among others, `isContract` will return false for the following
136     * types of addresses:
137     *
138     * - an externally-owned account
139     * - a contract in construction
140     * - an address where a contract will be created
141     * - an address where a contract lived, but was destroyed
142     *
143     * ====
144     */
145     function isContract(address account) internal view returns (bool) {
146         // This method relies in extcodesize, which returns 0 for contracts in
147         // construction, since the code is only stored at the end of the
148         // constructor execution.
149         uint256 size;

```

```

149 // solhint-disable-next-line no-inline-assembly
150 assembly {
151     assembly { size := extcodesize(account) }
152     return size > 0;
153 }
154 /**
155  * @dev Replacement for Solidity's `transfer`:
156  * sends `amount` wei to
157  * `recipient`, forwarding all available gas and
158  * reverting on errors.
159  *
160  * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
161  * of certain opcodes, possibly making contracts go over the 2300 gas limit
162  * imposed by `transfer`, making them unable to receive funds via
163  * `transfer`. {sendValue} removes this limitation.
164  *
165  * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
166  *
167  * IMPORTANT: because control is transferred to `recipient`, care must be
168  * taken to not create reentrancy vulnerabilities. Consider using
169  * {ReentrancyGuard}
170  */
171 function sendValue(address payable recipient, uint256 amount) internal {
172     require(address(this).balance >= amount, "Address: insufficient balance");
173
174     // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
175     (bool success, ) = recipient.call{ value: amount }("");
176     require(success, "Address: unable to send value, recipient may have reverted");
177 }
178 /**
179  * @dev Performs a Solidity function call using a low level `call`. A
180  * plain `call` is an unsafe replacement for a function call: use this
181  * function instead.
182  *
183  * If `target` reverts with a revert reason, it is bubbled up by this
184  * function (like regular Solidity function calls).
185  *
186  * Requirements:
187  *
188  * - `target` must be a contract.
189  * - calling `target` with `data` must not revert.
190  *
191  * _Available since v3.1._
192  */
193 function functionCall(address target, bytes memory data) internal returns (bytes memory) {
194     return functionCall(target, data, "Address: low-level call failed");

```

```

150 // solhint-disable-next-line no-inline-assembly
151 assembly {
152     assembly { size := extcodesize(account) }
153     return size > 0;
154 }
155 /**
156  * @dev Replacement for Solidity's `transfer`:
157  * sends `amount` wei to
158  * `recipient`, forwarding all available gas and
159  * reverting on errors.
160  *
161  * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
162  * of certain opcodes, possibly making contracts go over the 2300 gas limit
163  * imposed by `transfer`, making them unable to receive funds via
164  * `transfer`. {sendValue} removes this limitation.
165  *
166  * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
167  *
168  * IMPORTANT: because control is transferred to `recipient`, care must be
169  * taken to not create reentrancy vulnerabilities. Consider using
170  * {ReentrancyGuard}
171  */
172 function sendValue(address payable recipient, uint256 amount) internal {
173     require(address(this).balance >= amount, "Address: insufficient balance");
174
175     // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
176     (bool success, ) = recipient.call{ value: amount }("");
177     require(success, "Address: unable to send value, recipient may have reverted");
178 }
179 /**
180  * @dev Performs a Solidity function call using a low level `call`. A
181  * plain `call` is an unsafe replacement for a function call: use this
182  * function instead.
183  *
184  * If `target` reverts with a revert reason, it is bubbled up by this
185  * function (like regular Solidity function calls).
186  *
187  * Requirements:
188  *
189  * - `target` must be a contract.
190  * - calling `target` with `data` must not revert.
191  *
192  * _Available since v3.1._
193  */
194 function functionCall(address target, bytes memory data) internal returns (bytes memory) {
195     return functionCall(target, data, "Address: low-level call failed");

```

```

195     }
196
197     /**
198     * @dev Same as {xref-Address-functionCall-addr
ess-bytes-}[`functionCall`], but with
199     * `errorMessage` as a fallback revert reason w
hen `target` reverts.
200     *
201     * _Available since v3.1._
202     */
203     function functionCall(
204         address target,
205         bytes memory data,
206         string memory errorMessage
207     ) internal returns (bytes memory) {
208         return _functionCallWithValue(target, data,
0, errorMessage);
209     }
210
211     /**
212     * @dev Same as {xref-Address-functionCall-addr
ess-bytes-}[`functionCall`],
213     * but also transferring `value` wei to `target
`.
214     *
215     * Requirements:
216     *
217     * - the calling contract must have an ETH bala
nce of at least `value`.
218     * - the called Solidity function must be `paya
ble`.
219     *
220     * _Available since v3.1._
221     */
222     function functionCallWithValue(address target,
bytes memory data, uint256 value) internal returns
(bytes memory) {
223         return functionCallWithValue(target, data,
value, "Address: low-level call with value faile
d");
224     }
225
226     /**
227     * @dev Same as {xref-Address-functionCallWithV
alue-address-bytes-uint256-}[`functionCallWithValue
`], but
228     * with `errorMessage` as a fallback revert rea
son when `target` reverts.
229     *
230     * _Available since v3.1._
231     */
232     function functionCallWithValue(
233         address target,
234         bytes memory data,
235         uint256 value,
236         string memory errorMessage
237     ) internal returns (bytes memory) {
238         require(address(this).balance >= value, "Ad
dress: insufficient balance for call");
239         require(isContract(target), "Address: call
to non-contract");
240
241         // solhint-disable-next-line avoid-low-leve
l-calls
242         (bool success, bytes memory returndata) = t
arget.call{ value: value }(data);
243         return _verifyCallResult(success, returndat
a, errorMessage);

```

```

196     }
197
198     /**
199     * @dev Same as {xref-Address-functionCall-addr
ess-bytes-}[`functionCall`], but with
200     * `errorMessage` as a fallback revert reason w
hen `target` reverts.
201     *
202     * _Available since v3.1._
203     */
204     function functionCall(
205         address target,
206         bytes memory data,
207         string memory errorMessage
208     ) internal returns (bytes memory) {
209         return _functionCallWithValue(target, data,
0, errorMessage);
210     }
211
212     /**
213     * @dev Same as {xref-Address-functionCall-addr
ess-bytes-}[`functionCall`],
214     * but also transferring `value` wei to `target
`.
215     *
216     * Requirements:
217     *
218     * - the calling contract must have an ETH bala
nce of at least `value`.
219     * - the called Solidity function must be `paya
ble`.
220     *
221     * _Available since v3.1._
222     */
223     function functionCallWithValue(address target,
bytes memory data, uint256 value) internal returns
(bytes memory) {
224         return functionCallWithValue(target, data,
value, "Address: low-level call with value faile
d");
225     }
226
227     /**
228     * @dev Same as {xref-Address-functionCallWithV
alue-address-bytes-uint256-}[`functionCallWithValue
`], but
229     * with `errorMessage` as a fallback revert rea
son when `target` reverts.
230     *
231     * _Available since v3.1._
232     */
233     function functionCallWithValue(
234         address target,
235         bytes memory data,
236         uint256 value,
237         string memory errorMessage
238     ) internal returns (bytes memory) {
239         require(address(this).balance >= value, "Ad
dress: insufficient balance for call");
240         require(isContract(target), "Address: call
to non-contract");
241
242         // solhint-disable-next-line avoid-low-leve
l-calls
243         (bool success, bytes memory returndata) = t
arget.call{ value: value }(data);
244         return _verifyCallResult(success, returndat
a, errorMessage);

```

```

244     }
245
246     function _functionCallWithValue(
247         address target,
248         bytes memory data,
249         uint256 weiValue,
250         string memory errorMessage
251     ) private returns (bytes memory) {
252         require(isContract(target), "Address: call
to non-contract");
253
254         // solhint-disable-next-line avoid-low-level-calls
255         (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
256         if (success) {
257             return returndata;
258         } else {
259             // Look for revert reason and bubble it
up if present
260             if (returndata.length > 0) {
261                 // The easiest way to bubble the re
vert reason is using memory via assembly
262
263                 // solhint-disable-next-line no-inline-assembly
264                 assembly {
265                     let returndata_size := mload(re
turndata)
266                     revert(add(32, returndata), ret
urndata_size)
267                 }
268             } else {
269                 revert(errorMessage);
270             }
271         }
272     }
273
274     /**
275      * @dev Same as {xref-Address-functionCall-addr
ess-bytes-}[`functionCall`],
276      * but performing a static call.
277      *
278      * _Available since v3.3._
279      */
280     function functionStaticCall(address target, byt
es memory data) internal view returns (bytes memor
y) {
281         return functionStaticCall(target, data, "Ad
dress: low-level static call failed");
282     }
283
284     /**
285      * @dev Same as {xref-Address-functionCall-addr
ess-bytes-string-}[`functionCall`],
286      * but performing a static call.
287      *
288      * _Available since v3.3._
289      */
290     function functionStaticCall(
291         address target,
292         bytes memory data,
293         string memory errorMessage
294     ) internal view returns (bytes memory) {
295         require(isContract(target), "Address: stati
c call to non-contract");
296
297         // solhint-disable-next-line avoid-low-level-calls

```

```

245     }
246
247     function _functionCallWithValue(
248         address target,
249         bytes memory data,
250         uint256 weiValue,
251         string memory errorMessage
252     ) private returns (bytes memory) {
253         require(isContract(target), "Address: call
to non-contract");
254
255         // solhint-disable-next-line avoid-low-level-calls
256         (bool success, bytes memory returndata) = t
arget.call{ value: weiValue }(data);
257         if (success) {
258             return returndata;
259         } else {
260             // Look for revert reason and bubble it
up if present
261             if (returndata.length > 0) {
262                 // The easiest way to bubble the re
vert reason is using memory via assembly
263
264                 // solhint-disable-next-line no-inline-assembly
265                 assembly {
266                     let returndata_size := mload(re
turndata)
267                     revert(add(32, returndata), ret
urndata_size)
268                 }
269             } else {
270                 revert(errorMessage);
271             }
272         }
273     }
274
275     /**
276      * @dev Same as {xref-Address-functionCall-addr
ess-bytes-}[`functionCall`],
277      * but performing a static call.
278      *
279      * _Available since v3.3._
280      */
281     function functionStaticCall(address target, byt
es memory data) internal view returns (bytes memor
y) {
282         return functionStaticCall(target, data, "Ad
dress: low-level static call failed");
283     }
284
285     /**
286      * @dev Same as {xref-Address-functionCall-addr
ess-bytes-string-}[`functionCall`],
287      * but performing a static call.
288      *
289      * _Available since v3.3._
290      */
291     function functionStaticCall(
292         address target,
293         bytes memory data,
294         string memory errorMessage
295     ) internal view returns (bytes memory) {
296         require(isContract(target), "Address: stati
c call to non-contract");
297
298         // solhint-disable-next-line avoid-low-level-calls

```

```

298         (bool success, bytes memory returndata) = t
arget.staticcall(data);
299         return _verifyCallResult(success, returndata,
a, errorMessage);
300     }
301
302     /**
303      * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
304      * but performing a delegate call.
305      *
306      * _Available since v3.3._
307      */
308     function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory) {
309         return functionDelegateCall(target, data,
"Address: low-level delegate call failed");
310     }
311
312     /**
313      * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
314      * but performing a delegate call.
315      *
316      * _Available since v3.3._
317      */
318     function functionDelegateCall(
319         address target,
320         bytes memory data,
321         string memory errorMessage
322     ) internal returns (bytes memory) {
323         require(isContract(target), "Address: delegate call to non-contract");
324
325         // solhint-disable-next-line avoid-low-level-calls
326         (bool success, bytes memory returndata) = target.delegatecall(data);
327         return _verifyCallResult(success, returndata, errorMessage);
328     }
329
330     function _verifyCallResult(
331         bool success,
332         bytes memory returndata,
333         string memory errorMessage
334     ) private pure returns (bytes memory) {
335         if (success) {
336             return returndata;
337         } else {
338             // Look for revert reason and bubble it up if present
339             if (returndata.length > 0) {
340                 // The easiest way to bubble the revert reason is using memory via assembly
341
342                 // solhint-disable-next-line no-inline-assembly
343                 assembly {
344                     let returndata_size := mload(returndata)
345                     revert(add(32, returndata), returndata_size)
346                 }
347             } else {
348                 revert(errorMessage);
349             }
350         }
351     }

```

```

299         (bool success, bytes memory returndata) = target.staticcall(data);
300         return _verifyCallResult(success, returndata, errorMessage);
301     }
302
303     /**
304      * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
305      * but performing a delegate call.
306      *
307      * _Available since v3.3._
308      */
309     function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory) {
310         return functionDelegateCall(target, data, "Address: low-level delegate call failed");
311     }
312
313     /**
314      * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
315      * but performing a delegate call.
316      *
317      * _Available since v3.3._
318      */
319     function functionDelegateCall(
320         address target,
321         bytes memory data,
322         string memory errorMessage
323     ) internal returns (bytes memory) {
324         require(isContract(target), "Address: delegate call to non-contract");
325
326         // solhint-disable-next-line avoid-low-level-calls
327         (bool success, bytes memory returndata) = target.delegatecall(data);
328         return _verifyCallResult(success, returndata, errorMessage);
329     }
330
331     function _verifyCallResult(
332         bool success,
333         bytes memory returndata,
334         string memory errorMessage
335     ) private pure returns (bytes memory) {
336         if (success) {
337             return returndata;
338         } else {
339             // Look for revert reason and bubble it up if present
340             if (returndata.length > 0) {
341                 // The easiest way to bubble the revert reason is using memory via assembly
342
343                 // solhint-disable-next-line no-inline-assembly
344                 assembly {
345                     let returndata_size := mload(returndata)
346                     revert(add(32, returndata), returndata_size)
347                 }
348             } else {
349                 revert(errorMessage);
350             }
351         }
352     }

```

```

351     }
352
353     function addressToString(address _address) internal pure returns(string memory) {
354         bytes32 _bytes = bytes32(uint256(_address));
355         bytes memory HEX = "0123456789abcdef";
356         bytes memory _addr = new bytes(42);
357
358         _addr[0] = '0';
359         _addr[1] = 'x';
360
361         for(uint256 i = 0; i < 20; i++) {
362             _addr[2+i*2] = HEX[uint8(_bytes[i + 12]
363 >> 4)];
364             _addr[3+i*2] = HEX[uint8(_bytes[i + 12]
365 & 0x0f)];
366         }
367         return string(_addr);
368     }
369 }
370
371 library SafeERC20 {
372     using LowGasSafeMath for uint256;
373     using Address for address;
374
375     function safeTransfer(IERC20 token, address to,
376 uint256 value) internal {
377         _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
378     }
379
380     function safeTransferFrom(IERC20 token, address
381 from, address to, uint256 value) internal {
382         _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
383     }
384
385     /**
386      * @dev Deprecated. This function has issues similar to the ones found in
387      * {IERC20-approve}, and its usage is discouraged.
388      *
389      * Whenever possible, use {safeIncreaseAllowance} and
390      * {safeDecreaseAllowance} instead.
391      */
392     function safeApprove(IERC20 token, address spender, uint256 value) internal {
393         // safeApprove should only be called when setting an initial allowance,
394         // or when resetting it to zero. To increase and decrease it, use
395         // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
396         // solhint-disable-next-line max-line-length
397         require((value == 0) || (token.allowance(address(this), spender) == 0),
398             "SafeERC20: approve from non-zero to non-zero allowance");
399         _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
400     }
401

```

```

352     }
353
354     function addressToString(address _address) internal pure returns(string memory) {
355         bytes32 _bytes = bytes32(uint256(_address));
356         bytes memory HEX = "0123456789abcdef";
357         bytes memory _addr = new bytes(42);
358
359         _addr[0] = '0';
360         _addr[1] = 'x';
361
362         for(uint256 i = 0; i < 20; i++) {
363             _addr[2+i*2] = HEX[uint8(_bytes[i + 12]
364 >> 4)];
365             _addr[3+i*2] = HEX[uint8(_bytes[i + 12]
366 & 0x0f)];
367         }
368         return string(_addr);
369     }
370 }
371
372 library SafeERC20 {
373     using LowGasSafeMath for uint256;
374     using Address for address;
375
376     function safeTransfer(IERC20 token, address to,
377 uint256 value) internal {
378         _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
379     }
380
381     function safeTransferFrom(IERC20 token, address
382 from, address to, uint256 value) internal {
383         _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
384     }
385
386     /**
387      * @dev Deprecated. This function has issues similar to the ones found in
388      * {IERC20-approve}, and its usage is discouraged.
389      *
390      * Whenever possible, use {safeIncreaseAllowance} and
391      * {safeDecreaseAllowance} instead.
392      */
393     function safeApprove(IERC20 token, address spender, uint256 value) internal {
394         // safeApprove should only be called when setting an initial allowance,
395         // or when resetting it to zero. To increase and decrease it, use
396         // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
397         // solhint-disable-next-line max-line-length
398         require((value == 0) || (token.allowance(address(this), spender) == 0),
399             "SafeERC20: approve from non-zero to non-zero allowance");
400         _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
401     }
402

```



```

401     function safeIncreaseAllowance(IERC20 token, ad
dress spender, uint256 value) internal {
402         uint256 newAllowance = token.allowance(addr
ess(this), spender).add(value);
403         _callOptionalReturn(token, abi.encodeWithSe
lector(token.approve.selector, spender, newAllowanc
e));
404     }
405
406     function safeDecreaseAllowance(
407         IERC20 token,
408         address spender,
409         uint256 value
410     ) internal {
411         uint256 newAllowance = token.allowance(addr
ess(this), spender)
412             .sub(value);
413         _callOptionalReturn(token, abi.encodeWithSe
lector(token.approve.selector, spender, newAllowanc
e));
414     }
415
416     /**
417      * @dev Imitates a Solidity high-level call (i.
e. a regular function call to a contract), relaxing
the requirement
418      * on the return value: the return value is opt
ional (but if data is returned, it must not be fals
e).
419      * @param token The token targeted by the call.
420      * @param data The call data (encoded using ab
i.encode or one of its variants).
421      */
422     function _callOptionalReturn(IERC20 token, byte
s memory data) private {
423         // We need to perform a low level call her
e, to bypass Solidity's return data size checking m
echanism, since
424         // we're implementing it ourselves. We use
{Address.functionCall} to perform this call, which
verifies that
425         // the target address contains contract cod
e and also asserts for success in the low-level cal
l.
426
427         bytes memory returndata = address(token).fu
nctionCall(data, "SafeERC20: low-level call faile
d");
428         if (returndata.length > 0) { // Return data
is optional
429             // solhint-disable-next-line max-line-l
ength
430             require(abi.decode(returndata, (bool)),
"SafeERC20: ERC20 operation did not succeed");
431         }
432     }
433 }
434
435 contract OwnableData {
436     address public owner;
437     address public pendingOwner;
438 }
439
440 contract Ownable is OwnableData {
441     event OwnershipTransferred(address indexed prev
iousOwner, address indexed newOwner);
442
443     /// @notice `owner` defaults to msg.sender on c
onstruction.

```

```

402     function safeIncreaseAllowance(IERC20 token, ad
dress spender, uint256 value) internal {
403         uint256 newAllowance = token.allowance(addr
ess(this), spender).add(value);
404         _callOptionalReturn(token, abi.encodeWithSe
lector(token.approve.selector, spender, newAllowanc
e));
405     }
406
407     function safeDecreaseAllowance(
408         IERC20 token,
409         address spender,
410         uint256 value
411     ) internal {
412         uint256 newAllowance = token.allowance(addr
ess(this), spender)
413             .sub(value);
414         _callOptionalReturn(token, abi.encodeWithSe
lector(token.approve.selector, spender, newAllowanc
e));
415     }
416
417     /**
418      * @dev Imitates a Solidity high-level call (i.
e. a regular function call to a contract), relaxing
the requirement
419      * on the return value: the return value is opt
ional (but if data is returned, it must not be fals
e).
420      * @param token The token targeted by the call.
421      * @param data The call data (encoded using ab
i.encode or one of its variants).
422      */
423     function _callOptionalReturn(IERC20 token, byte
s memory data) private {
424         // We need to perform a low level call her
e, to bypass Solidity's return data size checking m
echanism, since
425         // we're implementing it ourselves. We use
{Address.functionCall} to perform this call, which
verifies that
426         // the target address contains contract cod
e and also asserts for success in the low-level cal
l.
427
428         bytes memory returndata = address(token).fu
nctionCall(data, "SafeERC20: low-level call faile
d");
429         if (returndata.length > 0) { // Return data
is optional
430             // solhint-disable-next-line max-line-l
ength
431             require(abi.decode(returndata, (bool)),
"SafeERC20: ERC20 operation did not succeed");
432         }
433     }
434 }
435
436 contract OwnableData {
437     address public owner;
438     address public pendingOwner;
439 }
440
441 contract Ownable is OwnableData {
442     event OwnershipTransferred(address indexed prev
iousOwner, address indexed newOwner);
443
444     /// @notice `owner` defaults to msg.sender on c
onstruction.

```

```

444     constructor() {
445         owner = msg.sender;
446         emit OwnershipTransferred(address(0), msg.s
ender);
447     }
448
449     /// @notice Transfers ownership to `newOwner`.
    Either directly or claimable by the new pending ow
ner.
450     /// Can only be invoked by the current `owner`.
451     /// @param newOwner Address of the new owner.
452     /// @param direct True if `newOwner` should be
    set immediately. False if `newOwner` needs to use
    `claimOwnership`.
453     /// @param renounce Allows the `newOwner` to be
    `address(0)` if `direct` and `renounce` is True. Ha
    s no effect otherwise.
454     function transferOwnership(
455         address newOwner,
456         bool direct,
457         bool renounce
458     ) public onlyOwner {
459         if (direct) {
460             // Checks
461             require(newOwner != address(0) || renou
nce, "Ownable: zero address");
462
463             // Effects
464             emit OwnershipTransferred(owner, newOwn
er);
465
466             owner = newOwner;
467             pendingOwner = address(0);
468         } else {
469             // Effects
470             pendingOwner = newOwner;
471         }
472     }
473
474     /// @notice Needs to be called by `pendingOwner`
    to claim ownership.
475     function claimOwnership() public {
476         address _pendingOwner = pendingOwner;
477
478         // Checks
479         require(msg.sender == _pendingOwner, "Ownab
le: caller != pending owner");
480
481         // Effects
482         emit OwnershipTransferred(owner, _pendingOw
ner);
483
484         owner = _pendingOwner;
485         pendingOwner = address(0);
486     }
487
488     /// @notice Only allows the `owner` to execute
    the function.
489     modifier onlyOwner() {
490         require(msg.sender == owner, "Ownable: call
er is not the owner");
491         _;
492     }
493
494     interface IMemo is IERC20 {
495         function rebase( uint256 ohmProfit_, uint epoch
_) external returns (uint256);
496
497         function circulatingSupply() external view retu
rns (uint256);

```

```

445     constructor() {
446         owner = msg.sender;
447         emit OwnershipTransferred(address(0), msg.s
ender);
448     }
449
450     /// @notice Transfers ownership to `newOwner`.
    Either directly or claimable by the new pending ow
ner.
451     /// Can only be invoked by the current `owner`.
452     /// @param newOwner Address of the new owner.
453     /// @param direct True if `newOwner` should be
    set immediately. False if `newOwner` needs to use
    `claimOwnership`.
454     /// @param renounce Allows the `newOwner` to be
    `address(0)` if `direct` and `renounce` is True. Ha
    s no effect otherwise.
455     function transferOwnership(
456         address newOwner,
457         bool direct,
458         bool renounce
459     ) public onlyOwner {
460         if (direct) {
461             // Checks
462             require(newOwner != address(0) || renou
nce, "Ownable: zero address");
463
464             // Effects
465             emit OwnershipTransferred(owner, newOwn
er);
466
467             owner = newOwner;
468             pendingOwner = address(0);
469         } else {
470             // Effects
471             pendingOwner = newOwner;
472         }
473     }
474
475     /// @notice Needs to be called by `pendingOwner`
    to claim ownership.
476     function claimOwnership() public {
477         address _pendingOwner = pendingOwner;
478
479         // Checks
480         require(msg.sender == _pendingOwner, "Ownab
le: caller != pending owner");
481
482         // Effects
483         emit OwnershipTransferred(owner, _pendingOw
ner);
484
485         owner = _pendingOwner;
486         pendingOwner = address(0);
487     }
488
489     /// @notice Only allows the `owner` to execute
    the function.
490     modifier onlyOwner() {
491         require(msg.sender == owner, "Ownable: call
er is not the owner");
492         _;
493     }
494
495     interface IMemo is IERC20 {
496         function rebase( uint256 ohmProfit_, uint epoch
_) external returns (uint256);
497
498         function circulatingSupply() external view retu
rns (uint256);

```

```

497
498     function balanceOf(address who) external view o
verride returns (uint256);
499
500     function gonsForBalance( uint amount ) external
view returns ( uint );
501
502     function balanceForGons( uint gons ) external v
iew returns ( uint );
503
504     function index() external view returns ( uint
);
505 }
506
507 interface IWarmup {
508     function retrieve( address staker_, uint amount
_ ) external;
509 }
510
511 interface IDistributor {
512     function distribute() external returns ( bool
);
513 }
514
515 contract TimeStaking is Ownable {
516
517     using LowGasSafeMath for uint256;
518     using LowGasSafeMath for uint32;
519     using SafeERC20 for IERC20;
520     using SafeERC20 for IMemo;
521
522     IERC20 public immutable Time;
523     IMemo public immutable Memories;
524
525     struct Epoch {
526         uint number;
527         uint distribute;
528         uint32 length;
529         uint32 endTime;
530     }
531     Epoch public epoch;
532
533     IDistributor public distributor;
534
535     uint public totalBonus;
536
537     IWarmup public warmupContract;
538     uint public warmupPeriod;
539
540     event LogStake(address indexed recipient, uint2
56 amount);
541     event LogClaim(address indexed recipient, uint2
56 amount);
542     event LogForfeit(address indexed recipient, uin
t256 memoAmount, uint256 timeAmount);
543     event LogDepositLock(address indexed user, bool
locked);
544     event LogUnstake(address indexed recipient, uin
t256 amount);
545     event LogRebase(uint256 distribute);
546     event LogSetContract(CONTRACTS contractType, ad
dress indexed _contract);
547     event LogWarmupPeriod(uint period);
548
549     constructor (
550         address _Time,
551         address _Memories,
552         uint32 _epochLength,
553         uint _firstEpochNumber,
554         uint32 _firstEpochTime

```

```

498
499     function balanceOf(address who) external view o
verride returns (uint256);
500
501     function gonsForBalance( uint amount ) external
view returns ( uint );
502
503     function balanceForGons( uint gons ) external v
iew returns ( uint );
504
505     function index() external view returns ( uint
);
506 }
507
508 interface IWarmup {
509     function retrieve( address staker_, uint amount
_ ) external;
510 }
511
512 interface IDistributor {
513     function distribute() external returns ( bool
);
514 }
515
516 contract TimeStaking is Ownable {
517
518     using LowGasSafeMath for uint256;
519     using LowGasSafeMath for uint32;
520     using SafeERC20 for IERC20;
521     using SafeERC20 for IMemo;
522
523     IERC20 public immutable Time;
524     IMemo public immutable Memories;
525
526     struct Epoch {
527         uint number;
528         uint distribute;
529         uint32 length;
530         uint32 endTime;
531     }
532     Epoch public epoch;
533
534     IDistributor public distributor;
535
536     uint public totalBonus;
537
538     IWarmup public warmupContract;
539     uint public warmupPeriod;
540
541     event LogStake(address indexed recipient, uint2
56 amount);
542     event LogClaim(address indexed recipient, uint2
56 amount);
543     event LogForfeit(address indexed recipient, uin
t256 memoAmount, uint256 timeAmount);
544     event LogDepositLock(address indexed user, bool
locked);
545     event LogUnstake(address indexed recipient, uin
t256 amount);
546     event LogRebase(uint256 distribute);
547     event LogSetContract(CONTRACTS contractType, ad
dress indexed _contract);
548     event LogWarmupPeriod(uint period);
549
550     constructor (
551         address _Time,
552         address _Memories,
553         uint32 _epochLength,
554         uint _firstEpochNumber,
555         uint32 _firstEpochTime

```

```

555     ) {
556         require( _Time != address(0) );
557         Time = IERC20(_Time);
558         require( _Memories != address(0) );
559         Memories = IMemo(_Memories);
560
561         epoch = Epoch({
562             length: _epochLength,
563             number: _firstEpochNumber,
564             endTime: _firstEpochTime,
565             distribute: 0
566         });
567     }
568
569     struct Claim {
570         uint deposit;
571         uint gons;
572         uint expiry;
573         bool lock; // prevents malicious delays
574     }
575     mapping( address => Claim ) public warmupInfo;
576
577     /**
578      * @notice stake Time to enter warmup
579      * @param _amount uint
580      * @return bool
581      */
582     function stake( uint _amount, address _recipien
583         t ) external returns ( bool ) {
584         rebase();
585
586         Time.safeTransferFrom( msg.sender, address
587             (this), _amount );
588
589         Claim memory info = warmupInfo[ _recipient
590             ];
591         require( !info.lock, "Deposits for account
592             are locked" );
593
594         warmupInfo[ _recipient ] = Claim ({
595             deposit: info.deposit.add( _amount ),
596             gons: info.gons.add( Memories.gonsForBa
597                 lance( _amount ) ),
598             expiry: epoch.number.add( warmupPeriod
599             ),
600             lock: false
601         });
602
603         Memories.safeTransfer( address(warmupContra
604             ct), _amount );
605         emit LogStake(_recipient, _amount);
606         return true;
607     }
608
609     /**
610      * @notice retrieve MEMO from warmup
611      * @param _recipient address
612      */
613     function claim ( address _recipient ) external
614         {
615         Claim memory info = warmupInfo[ _recipient
616             ];
617         if ( epoch.number >= info.expiry && info.ex
618             piry != 0 ) {
619             delete warmupInfo[ _recipient ];
620             uint256 amount = Memories.balanceForGon
621                 s( info.gons );

```

```

556     ) {
557         require( _Time != address(0) );
558         Time = IERC20(_Time);
559         require( _Memories != address(0) );
560         Memories = IMemo(_Memories);
561
562         epoch = Epoch({
563             length: _epochLength,
564             number: _firstEpochNumber,
565             endTime: _firstEpochTime,
566             distribute: 0
567         });
568     }
569
570     struct Claim {
571         uint deposit;
572         uint gons;
573         uint expiry;
574         bool lock; // prevents malicious delays
575     }
576     mapping( address => Claim ) public warmupInfo;
577
578     /**
579      * @notice stake Time to enter warmup
580      * @param _amount uint
581      * @return bool
582      */
583     function stake( uint _amount, address _recipien
584         t ) external returns ( bool ) {
585         rebase();
586
587         Time.safeTransferFrom( msg.sender, address
588             (this), _amount );
589
590         Claim memory info = warmupInfo[ _recipient
591             ];
592         require( !info.lock, "Deposits for account
593             are locked" );
594
595         warmupInfo[ _recipient ] = Claim ({
596             deposit: info.deposit.add( _amount ),
597             gons: info.gons.add( Memories.gonsForBa
598                 lance( _amount ) ),
599             expiry: epoch.number.add( warmupPeriod
600             ),
601             lock: false
602         });
603
604         Memories.safeTransfer( address(warmupContra
605             ct), _amount );
606         emit LogStake(_recipient, _amount);
607         return true;
608     }
609
610     /**
611      * @notice retrieve MEMO from warmup
612      * @param _recipient address
613      */
614     function claim ( address _recipient ) external
615         {
616         Claim memory info = warmupInfo[ _recipient
617             ];
618         if ( epoch.number >= info.expiry && info.ex
619             piry != 0 ) {
620             delete warmupInfo[ _recipient ];
621             uint256 amount = Memories.balanceForGon
622                 s( info.gons );

```

```

611         warmupContract.retrieve( _recipient, a
mount);
612         emit LogClaim(_recipient, amount);
613     }
614 }
615
616 /**
617     @notice forfeit MEMO in warmup and retrieve
Time
618     */
619     function forfeit() external {
620         Claim memory info = warmupInfo[ msg.sender
];
621         delete warmupInfo[ msg.sender ];
622         uint memoBalance = Memories.balanceForGons(
info.gons );
623         warmupContract.retrieve( address(this), me
moBalance);
624         Time.safeTransfer( msg.sender, info.deposi
t);
625         emit LogForfeit(msg.sender, memoBalance, in
fo.deposit);
626     }
627
628 /**
629     @notice prevent new deposits to address (pr
otection from malicious activity)
630     */
631     function toggleDepositLock() external {
632         warmupInfo[ msg.sender ].lock = !warmupInfo
[ msg.sender ].lock;
633         emit LogDepositLock(msg.sender, warmupInfo[
msg.sender ].lock);
634     }
635
636 /**
637     @notice redeem MEMO for Time
638     @param _amount uint
639     @param _trigger bool
640     */
641     function unstake( uint _amount, bool _trigger )
external {
642         if ( _trigger ) {
643             rebase();
644         }
645         Memories.safeTransferFrom( msg.sender, addr
ess(this), _amount );
646         Time.safeTransfer( msg.sender, _amount );
647         emit LogUnstake(msg.sender, _amount);
648     }
649
650 /**
651     @notice returns the MEMO index, which track
s rebase growth
652     @return uint
653     */
654     function index() external view returns ( uint )
{
655         return Memories.index();
656     }
657
658 /**
659     @notice trigger rebase if epoch over
660     */
661     function rebase() public {
662         if( epoch.endTime <= uint32(block.timestam
p) ) {
663

```

```

612         warmupContract.retrieve( _recipient, a
mount);
613         emit LogClaim(_recipient, amount);
614     }
615 }
616
617 /**
618     @notice forfeit MEMO in warmup and retrieve
Time
619     */
620     function forfeit() external {
621         Claim memory info = warmupInfo[ msg.sender
];
622         delete warmupInfo[ msg.sender ];
623         uint memoBalance = Memories.balanceForGons(
info.gons );
624         warmupContract.retrieve( address(this), me
moBalance);
625         Time.safeTransfer( msg.sender, info.deposi
t);
626         emit LogForfeit(msg.sender, memoBalance, in
fo.deposit);
627     }
628
629 /**
630     @notice prevent new deposits to address (pr
otection from malicious activity)
631     */
632     function toggleDepositLock() external {
633         warmupInfo[ msg.sender ].lock = !warmupInfo
[ msg.sender ].lock;
634         emit LogDepositLock(msg.sender, warmupInfo[
msg.sender ].lock);
635     }
636
637 /**
638     @notice redeem MEMO for Time
639     @param _amount uint
640     @param _trigger bool
641     */
642     function unstake( uint _amount, bool _trigger )
external {
643         if ( _trigger ) {
644             rebase();
645         }
646         Memories.safeTransferFrom( msg.sender, addr
ess(this), _amount );
647         Time.safeTransfer( msg.sender, _amount );
648         emit LogUnstake(msg.sender, _amount);
649     }
650
651 /**
652     @notice returns the MEMO index, which track
s rebase growth
653     @return uint
654     */
655     function index() external view returns ( uint )
{
656         return Memories.index();
657     }
658
659 /**
660     @notice trigger rebase if epoch over
661     */
662     function rebase() public {
663         if( epoch.endTime <= uint32(block.timestam
p) ) {
664

```

```

664         Memories.rebase( epoch.distribute, epoc
h.number );
665
666         epoch.endTime = epoch.endTime.add32( ep
och.length );
667         epoch.number++;
668
669         if ( address(distributor) != address(0)
) {
670             distributor.distribute();
671         }
672
673         uint balance = contractBalance();
674         uint staked = Memories.circulatingSuppl
y();
675
676         if( balance <= staked ) {
677             epoch.distribute = 0;
678         } else {
679             epoch.distribute = balance.sub( sta
ked );
680         }
681         emit LogRebase(epoch.distribute);
682     }
683 }
684
685 /**
686  @notice returns contract Time holdings, inc
luding bonuses provided
687  @return uint
688  */
689     function contractBalance() public view returns
( uint ) {
690         return Time.balanceOf( address(this) ).add(
totalBonus );
691     }
692
693     enum CONTRACTS { DISTRIBUTOR, WARMUP }
694
695     /**
696     @notice sets the contract address for LP st
aking
697     @param _contract address
698     */
699     function setContract( CONTRACTS _contract, addr
ess _address ) external onlyOwner {
700         if( _contract == CONTRACTS.DISTRIBUTOR ) {
701             // 0
702             distributor = IDistributor(_address);
703         } else if ( _contract == CONTRACTS.WARMUP )
{ // 1
704             require( address(warmupContract) == add
ress( 0 ), "Warmup cannot be set more than once" );
705             warmupContract = IWarmup(_address);
706         }
707         emit LogSetContract(_contract, _address);
708     }
709
710     /**
711     * @notice set warmup period in epoch's numbers
for new stakers
712     * @param _warmupPeriod uint
713     */
714     function setWarmup( uint _warmupPeriod ) extern
al onlyOwner {
715         warmupPeriod = _warmupPeriod;
716         emit LogWarmupPeriod(_warmupPeriod);
717     }
718 }

```

```

665         Memories.rebase( epoch.distribute, epoc
h.number );
666
667         epoch.endTime = epoch.endTime.add32( ep
och.length );
668         epoch.number++;
669
670         if ( address(distributor) != address(0)
) {
671             distributor.distribute();
672         }
673
674         uint balance = contractBalance();
675         uint staked = Memories.circulatingSuppl
y();
676
677         if( balance <= staked ) {
678             epoch.distribute = 0;
679         } else {
680             epoch.distribute = balance.sub( sta
ked );
681         }
682         emit LogRebase(epoch.distribute);
683     }
684 }
685
686 /**
687  @notice returns contract Time holdings, inc
luding bonuses provided
688  @return uint
689  */
690     function contractBalance() public view returns
( uint ) {
691         return Time.balanceOf( address(this) ).add(
totalBonus );
692     }
693
694     enum CONTRACTS { DISTRIBUTOR, WARMUP }
695
696     /**
697     @notice sets the contract address for LP st
aking
698     @param _contract address
699     */
700     function setContract( CONTRACTS _contract, addr
ess _address ) external onlyOwner {
701         if( _contract == CONTRACTS.DISTRIBUTOR ) {
702             // 0
703             distributor = IDistributor(_address);
704         } else if ( _contract == CONTRACTS.WARMUP )
{ // 1
705             require( address(warmupContract) == add
ress( 0 ), "Warmup cannot be set more than once" );
706             warmupContract = IWarmup(_address);
707         }
708         emit LogSetContract(_contract, _address);
709     }
710
711     /**
712     * @notice set warmup period in epoch's numbers
for new stakers
713     * @param _warmupPeriod uint
714     */
715     function setWarmup( uint _warmupPeriod ) extern
al onlyOwner {
716         warmupPeriod = _warmupPeriod;
717         emit LogWarmupPeriod(_warmupPeriod);
718     }

```

