GET TOKEN PUBLIC EXCHANGE AGREEMENT

By performing the Exchange during the Exchange Period, you agree to become party to this Agreement and you agree that your counterparty is the Foundation.

1 DEFINITIONS

"**Activation Threshold**" has the meaning set out in para. 4.4;

"**Agreement**" means this GET Token Public Exchange Agreement and its annexes;

"**ETH Token**" means an Ether token created by Stiftung Ethereum;

"**Exchange**" means the transfer of ETH Tokens to the Smart Contract System and the Smart Contract System creating GET and transferring to you GET Tokens against the Exchange Rate during the Exchange Period;

"**Exchange Period**" has the meaning set out in para. 4.2;

"**Exchange Rate**" has the meaning set out in para. 4.5;

"**Foundation**" means Stichting GET-Protocol, a foundation incorporated under the laws of the Netherlands (*stichting*) having its seat at Jan Tooropstraat 1, (1062BK) Amsterdam, registered with the Trade Register with number 69771138;

"**GET-Protocol**" means the open source suite of smart contracts on the Ethereum Blockchain that will be developed by the Foundation to enable secure ticketing sales via the blockchain and that may also include additional functionalities;

"**GET Token**" means the ERC20 token that will be created by the Foundation through the Smart Contract System.

"**Guaranteed Exchange Rate**" has the meaning set out in para. 4.9;

"**Marketing Pool**" means the fund described in chapter 8;

"**Maximum Token Quantity**" has the meaning set out in para. 3.1;

"**Party**" or "**Parties**" means the parties that perform the Exchange: you and the Foundation;

"**Smart Contract System**" means the software code existing on the Ethereum Blockchain, a plaintext copy of which is attached as Annex 1, which governs the Exchange;

"**Stability Fund** " means the fund described in chapter 6;

"**Team and Advisors Pool**" means the fund described in chapter 7;

"**User Growth Fund**" means the fund described in chapter 5; NS

"**Validation Period**" has the meaning set out in para. 4.7.

2 PRINCIPLES

2.1 You understand and accept that the ETH Tokens you exchange for GET Tokens will be fully owned by the Foundation and that the Foundation will use the ETH Tokens for the development of the GET-Protocol. You understand and accept that the information contained on the internet about the GET-Protocol, including blogs and the white paper, are of descriptive nature only, that such information therefore is not binding between the Parties and thus do – unless explicitly referred to herein – not form part of this Agreement.

2.2 You understand and accept that while the Foundation will make reasonable efforts to develop the GET-Protocol, it is possible that such development may fail and that GET Tokens may become useless and/or valueless for technical, commercial, regulatory or other reasons. You are also aware of the risk that, even if all or parts of the GET-Protocol are successfully developed and released in full or in parts, due to a lack of interest from target stakeholders, the GET-Protocol could be fully or partially abandoned, remain commercially unsuccessful or shut down for lack of interest or other reasons**.** You therefore understand and accept that the Exchange carries significant financial, regulatory, reputational and other risks (see further chapter 12 regarding Risks).

2.3 You furthermore understand and accept that the Exchange is smart contract based and that by entering into the Exchange, you expressly agree to all of the terms and conditions set forth in Smart Contract System, consisting of software code, existing on the Ethereum blockchain, a plaintext copy of

which is attached to this Agreement as Annex 1. To the extent this Agreement contradicts the code of the Smart Contract System, the code of the Smart Contract System prevails. You hereby confirm to have carefully reviewed the Smart Contract System code, its functions and this Agreement, and that you understand the risks and costs of the Exchange.

2.4 This Agreement concerns the exchange of goods (*goederen*) that are not securities, debts, financial products or financial instruments nor qualify as (electronic) money. This Agreement therefore does not constitute a prospectus of any sort, is not a solicitation for investment and to our knowledge does not pertain in any way to an offering of securities or other financial products or instruments falling under the scope of financial regulations in any jurisdiction. Rather, this Agreement (and its Annexes) describe the properties and functionalities of the GET Token you will own after performing the Exchange.

2.5 By performing the Exchange, no form of ownership, partnership, joint venture or any similar relationship between you, the Foundation or any other individuals or entities involved with the deployment of the GET-Protocol and the Smart Contract System is created. You also acknowledge that GET Tokens do not confer unto you any economic or control rights over the GET Foundation, the GET-Protocol or the Smart Contract System.

2.6 You acknowledge that the Foundation has no obligation to buy back any GET Tokens from you, is not indebted to you, otherwise required to repay you in money or in kind, provide any services or to deliver any goods, products or property rights to you as a result of the Exchange and this Agreement.

3 GET Token creation and exchange

3.1 Maximum Token Quantity: During the Creation Period, up to 90 million GET Tokens (the "**Maximum Token Quantity**") will be created by the Foundation via the Smart Contract System.

3.2 Pools: The Maximum Token Quantity will be divided by the Smart Contract System into five different pools:

- *Pool A* consisting of up to 36,900,000 GET Tokens (the "**Pool A Maximum**"), which equals 41% of the Maximum Token Quantity, intended to be created by the Smart Contract System during the Exchange Period and to be exchanged against ETH Tokens (see chapter 4 below);
- *Pool B* consisting of up to 27,000,000 GET Tokens (the "**Pool B Maximum**"), which equals 30% of the Maximum Token Quantity, intended to be created by the Smart Contract System at the end of the Exchange Period, to be allocated to the User Growth Fund (see chapter 5 below);
- *Pool C* consisting of 12,600,000 GET Tokens in total, which equals 14% of the Maximum Token Quantity, intended to be created by the Smart Contract System at the end of the Exchange Period, to be allocated to the Stability Fund (see chapter 6 below);
- *Pool D* consisting of up to 11,700,000 GET Token (the "**Pool D Maximum**"), which equals 13% of the Maximum Token Quantity, intended to be created by the Smart Contract System at the end of the Exchange Period, to be allocated to (future) advisors and team members (see chapter 7 below); and
- *Pool E* consisting of 1,800,000 GET Tokens, which equals of 2% of the Maximum Token Quantity, intended to be created by the Smart Contract System at the end of the Exchange Period, to be allocated to the Marketing Fund (see chapter 8 below).

3.3 Dynamic and static pools: The amount of GET Tokens allocated to Pool C and Pool E are fixed (static) while the amount of GET Tokens to be allocated Pool A, Pool B and Pool D are dynamic, the size of Pool A depending on how many persons perform the Exchange during the Exchange period and the size of Pool B and Pool D to be calculated using the following formula: *("GET Tokens exchanged during the Exchange Period" / "Pool A Maximum") times "Pool [B OR C] Maximum"* as further clarified in the table below:

% of Pool A exchanged	Size Pool A	Size Pool B	Size Pool C	Size Pool D	Size Pool E	Total
100%	36,900,000	27,000,000	12,600,000	11,700,000	1,800,000.00	90,000,000
90%	33,210,000	24,300,000	12,600,000	10,530,000	1,620,000.00	82,440,000

80%	29,520,000	21,600,000	12,600,000	9,360,000	1,440,000.00	74,880,000
60%	22,140,000	16,200,000	12,600,000	7,020,000	1,080,000.00	59,760,000
50%	18,450,000	13,500,000	12,600,000	5,850,000	900,000.00	52,200,000
40%	14,760,000	10,800,000	12,600,000	4,680,000	720,000.00	44,640,000
30%	11,070,000	8,100,000	12,600,000	3,510,000	540,000.00	37,080,000
23% (Activation Threshold)	8,560,800	6,264,000	12,600,000	2,714,400	417,600.00	31,939,200
<23%	0	0	0	0	0	0

3.4 GET Token Creation Cap: No additional GET Tokens will be created after the end of the Validation Period.

4 Pool A – "ICO" exchange

4.1 Pool A Maximum: up to 36,900,000 GET Tokens, which equals 41% of the Maximum Token Quantity, may be created by the Smart Contract System during the Exchange Period in exchange for ETH Tokens.

4.2 Exchange Period: The Exchange Period starts on 15 November 2017 on 13:00 CET and lasts until the Pool A Maximum has been exchanged or up until 15 December 2017 on 13:00 CET, whichever is earlier.

4.3 GET Token Creation: The creation and exchange of GET Tokens by the Smart Contract System during the Exchange Period are initiated by you sending an amount of ETH Tokens to the Smart Contract System, located on the Ethereum blockchain at the addresses set forth under para. 2.3. The Smart Contract System will create and allocate the corresponding GET Tokens to the wallet address the ETH Tokens were sent from, directly after the Activation Threshold has been met during the Exchange Period. Issuance of GET Tokens (meaning that respective entries of your Public Key (PUK) of User) are made by the Smart Contract System on the blockchain according to the allocation information in the Smart Contract System.

4.4 Activation Threshold: If less than 8,560,800 GET Tokens allocated to Pool A are exchanged during the Exchange Period, the Activation Threshold is not met, meaning that the Smart Contract System will automatically transfer back your ETH Tokens to the address used to transfer the ETH Tokens to the Smart Contract System. As soon as the Activation Threshold is met, the Smart Contract System will transfer GET Tokens to anyone who has or will perform an Exchange. GET Tokens will be transferred to the address from which ETH Tokens were received.

4.5 Exchange Rate: During the Public Exchange Period the Exchange Rate will be as follows:

GET Pool A Tokens	Name	ETH Token exchange rated in EUR (approx.)
0 to 1,365,300	Tier 0	EUR 0.37
1,365,301 to 6,789,600	Tier 1	EUR 0.42
6,789,601 to 13,579,200	Tier 2	EUR 0.44
13,579,201 to 20,959,200	Tier 3	EUR 0.47
20,959,201 and further	Tier 4	EUR 0.49

The value of ETH Tokens in euro has been determined on the basis of the exchange rate at 14 November 2017 13:00 (CET) as found on http://ethereumaverage.com/en/ethereum-price/eth-to-usd.

4.6 Value Added Tax: The Exchange to the best of our knowledge is exempted from VAT. Should tax authorities rule otherwise the Exchange Rate will be deemed to be inclusive of VAT.

4.7 Validation Period: The GET Tokens are coded as such that you cannot transfer them to another address until 13.00 CET on the 20th day following the date on which the Exchange Period ended (i.e. if

the Exchange Period ends on 15 December 2017 the GET Tokens can be transferred as of 2 January 2018, 13:00 CET), a period which the Foundation will use to validate all Exchanges.

4.8 Cancellation: If you qualify as a consumer pursuant to Dutch law (a person performing the Exchange outside your profession or trade) you have, pursuant to Dutch law, the right to cancel the Exchange within 14 days after you have received the GET Tokens, meaning that by sending an e-mail within that period to the e-mail address set out in para. 14.1, in which you declare that you wish to cancel the Exchange and provide the public address from which you transferred the ETH Tokens to the Smart Contract System, and the amount of ETH Tokens you transferred, the Foundation will transfer back the ETH Tokens to that public address at the end of the period mentioned in para. 4.7, provided it can validate that you are counterparty to this Agreement and that you are a consumer, minus associated Ethereum gas costs.

4.9 Guaranteed Exchange Rate: Once the GET-Protocol is sufficiently developed to allow third parties to sell tickets via the GET-Protocol, the Stability Fund will, when such third party (i.e. an event organizer) requires GET Tokens to sell tickets using the GET-Protocol, offer to exchange certain quantities of GET Tokens back to ETH Tokens on the open market, the (fractional) amount of ETH Tokens being offered in exchange for a GET Token being EUR 0.50 at minimum (i.e. if the average value of an ETH Token on the market is EUR 200, the Stability Fund will offer to exchange 0.0025 ETH Token for a GET Token). The Foundation guarantees that the Stability Fund will not offer to exchange against a lower exchange rate (the "**Guaranteed Exchange Rate**"). You understand and accept that the Foundation explicitly does not have any obligation to exchange GET Tokens back into ETH Tokens and therefore does not and cannot guarantee that you can exchange GET Tokens back with the Foundation, primarily for reasons set out in para. 2.2.

4.10 Functionalities of GET Tokens: All GET Tokens have the same functionalities, however, only GET Tokens sent from addresses that are "whitelisted" by the Foundation may interact with the GET-Protocol. The Foundation will only "whitelist" addresses that it has created itself for parties such as event organisers, ticketing companies and persons that purchase tickets via the GET-Protocol. Such accounts will not accept transfers of GET Tokens from accounts that were not created by the Foundation. Accordingly, you cannot transfer your GET Tokens to such accounts (wallets) within the GET-Protocol. An important reason for this policy are current regulatory uncertainties. In the future the Foundation intends to change this policy, meaning that you, for example, could use GET Tokens obtained as a result of the Exchange to buy tickets. You, however, accept and understand that the Foundation has no obligation to do so and does and cannot not guarantee that you will be given the rights to do so.

4.11 No other rights: You understand and accept that GET Tokens do not represent or constitute any ownership right or stake, share or security or equivalent rights or any right to receive future revenues, shares, property or intellectual property rights or any other form of participation in or relating to the GET-Protocol and/or the Foundation.

5 Pool B – USER GROWTH FUND

5.1 Pool B Maximum: In accordance with para. 3.3, up to 27,000,000 GET Tokens, which equals 30% of the Maximum Token Quantity, will at the end of the Exchange Period be directly released by the Smart Contract System to the User Growth Fund, which is controlled by the Foundation. Until the GET-Protocol is sufficiently developed, these GET Tokens will be transferred to an account for which only the Foundation will hold the private key.

5.2 Use: Once the GET-Protocol has, in the Foundation's reasonable opinion, been sufficiently developed, the GET Tokens allocated to the User Growth Fund will be used, at the sole discretion of the Foundation, to promote adoption of the GET-Protocol by stakeholders, for example, by lowering the amount of GET Tokens required to create an event or sell/buy tickets or to subsidize the secondary sale of tickets via the GET-Protocol. Such (residual) GET Tokens will be transferred to accounts of these stakeholders created by the Foundation and for which the Foundation only holds the private key. Such GET Tokens can only be transferred to addresses that are "whitelisted" by the Foundation. The Foundation will use reasonable efforts to ensure that these GET Tokens cannot be exchanged for other cryptocurrencies or fiat money on the open market. In the future the Foundation may change

this policy, more specifically when Pool A GET Tokens would be allowed to fully interact with the GET-Protocol.

6 Pool C – Stability fund

6.1 Pool C Maximum: In accordance with para. 3.3, 12,600,000 GET Tokens, which equals 14% of the Maximum Token Quantity, will at the end of the Exchange Period be directly released by the Smart Contract System to the Stability Fund. Until the GET-Protocol is sufficiently developed, these GET Tokens will be placed in escrow on an account for which only the Foundation will hold the private key.

6.2 Use: Once the GET-Protocol has been sufficiently developed, the GET Tokens allocated to the Stability Fund will be used to provide event organisers, ticketing companies and ticket buyers with the GET Tokens required to interact with the GET-Protocol against stable exchange rates. After such exchange, the Stability Fund will open an exchange contract on the open market offering to exchange ETH Tokens into GET Tokens against at least the Guaranteed Exchange Rate so that the Stability Fund is replenished to the extent that it again holds 12,600,000 GET Tokens.

7 Pool D – team and advisors pool

7.1 Pool D Maximum: In accordance with para. 3.3, up to 11,700,000 GET Tokens, which equals 13% of the Maximum Token Quantity, will be allocated to current and future advisors and team members. At the end of the Exchange Period, 5,535,000 GET Tokens (6.25%) will be directly released by the Smart Contract System to current advisors and team members. This number may vary over time within reasonable limits, due to the fact advisors are still being on boarded. The remainder of the GET Tokens that may be allocated to Pool D in accordance with para. 3.3 may, at the sole discretion of the Foundation, be exchanged with future team members and advisors if the Foundation deems this beneficial to the development and success of the GET-Protocol.

7.2 Transferability: The GET Tokens are coded as such that they cannot be moved to another account for a period of 12 months following the Exchange Period.

8 Pool E – Marketing pool

8.1 Pool E Maximum: In accordance with para. 3.3, 1,800,000 GET Tokens, which equals 2% of the Maximum Token Quantity, will at the end of the Exchange Period be directly released by the Smart Contract System to the Marketing Fund, which is an address for which the Foundation holds the private key.

8.2 Use: The GET Tokens will, at the sole discretion of the Foundation, be transferred within a certain period after the Validation Period to third parties in exchange for services in the field of marketing and as a bounty rewards for quality control for the integrity and robustness of the code of the GET-Protocol. Such third parties are allowed to exchange these GET Tokens on the open market.

9 representations and warranties

9.1 By performing the Exchange you represent and warrant that:

- you have a deep understanding of the functionality, usage, storage, transmission mechanisms and intricacies associated with cryptographic tokens, like bitcoin (BTC) and Ether (ETH), and blockchain-based software systems;
- you have carefully reviewed this Agreement and the code of the Smart Contract System located on the Ethereum blockchain at the addresses set forth under para 2.3 and fully understand and accept the functions implemented therein;
- you are legally permitted to transfer ETH Tokens to the Smart Contract System in exchange for GET Tokens in your jurisdiction;
- you confirm that your financial position (*draagkracht*) is such that you can perform the Exchange and bear associated risks, including complete loss of value, and still perform your other financial obligations;
- you are of a sufficient age to legally obtain GET Tokens;
- you will take sole responsibility for any restrictions and risks associated with GET Tokens;

- you are not submitting ETH Tokens to the Smart Contract System to obtain GET Tokens for the purpose of speculative investment;
- you are not obtaining or using GET Tokens for any illegal purposes;
- you waive the right to participate in a class action lawsuit or a class wide arbitration against any entity or individual involved with the creation of GET Tokens;
- you understand the Exchange does not involve the purchase of shares or any equivalent in any existing or future public or private company, corporation or other entity in any jurisdiction;
- you understand the Exchange does not confer unto you any economic or control rights over the GET-Protocol, the Smart Contract System or the Foundation;
- you understand that the Foundation has no obligation to buy back any GET Tokens from you, is indebted to you, otherwise required to repay you in money or in kind, provide any services or to deliver any goods, products or (intellectual) property rights to you as a result of the Exchange and this Agreement;
- you understand that the Exchange carries significant financial, regulatory and reputational risks;
- you understand that the GET Tokens have the characteristics (*eigenschappen*) as set forth in this Agreement and that no additional (implied) rights or characteristics may be derived from any other documentation or communication with regard to GET Tokens, the GET-Protocol, the Smart Contract System and the Foundation;
- you have the full title to the ETH Tokens you exchange and they are not encumbered;
- you understand and expressly accept that there is no warranty whatsoever on GET Tokens, the Smart Contract System and/or the success of the GET-Protocol, expressed or implied, to the extent permitted by law, and that the Smart Contract System is used and GET Tokens are obtained at your sole risk on an "as is" and "under development" basis and without, to the extent permitted by law, any warranties of any kind, including, but not limited to, warranties of title or implied warranties, merchantability or fitness for a particular purpose;
- you understand that you have, other than as implemented in the Smart Contract System in case the Activation Threshold has not been reached (see para 4.4) and other than pursuant para. 4.8 (cancellation), and to the extent permitted by law, no right to request any refund or transfer back of the ETH Tokens submitted to the Smart Contract System for the Exchange under any circumstance;
- you understand with regard to GET Tokens no market liquidity may be guaranteed and that the value of GET Tokens over time may experience extreme volatility or depreciate in full; and
- you understand that you bear the sole responsibility to determine the Exchange, the potential appreciation or depreciation in the value of GET Tokens over time, the exchange of GET Tokens and/or any other action or transaction related to the GET-Protocol have tax implications for you; by exchanging, holding or using GET Tokens, and to the extent permitted by law, you agree not to hold the Foundation or any third party liable for any tax liability associated with or arising from the exchange, ownership or use of GET Tokens or any other action or transaction related to the GET-Protocol.

10 GET-PROTOCOL PROJECT EXECUTION

10.1 You understand and accept that the development and execution of the GET-Protocol will be undertaken by the Foundation. The Foundation may engage subcontractors to perform the entire or partial development and execution of the GET-Protocol. The scope of the development work will be triggered by the amount of GET Tokens exchanged for ETH Tokens during the Exchange Period. You understand that ETH Tokens owned by the Foundation as a result of the Exchange will be used for development and operation of the GET-Protocol.

10.2 You understand and accept that you have no influence over governance of the GET-Protocol or the Foundation.

10.3 You understand and accept that the GET-Protocol will need to go through substantial development works as part of which it may become subject of significant conceptual, technical and commercial changes before release. You understand and accept that as part of the development, an upgrade of the GET Token or GET-Protocol may be required, such decision to be made at the sole discretion of the Foundation (hard-fork of the GET Token or deployment of the GET-Protocol to a different blockchain), and that, if you decide not to participate in such an upgrade, you may no longer use your GET Token in the GET-Protocol and that non-upgraded GET Tokens may lose their functionality in full.

11 AUDIT OF THE SMART CONTRACT SYSTEM

11.1 The Foundation made reasonable efforts to have the Smart Contract System and GET-Protocol audited and approved by legal and technical experts during development. However, you understand and accept that smart contract technology is still in an early development stage and its application of experimental nature which carries significant operational, technological, financial, regulatory and reputational risks. Accordingly, while audits should raise the level of security and accuracy, you understand and accept that audits do not amount to any form of warranty, including direct or indirect warranties that the Smart Contract System, GET-Protocol and GET Tokens will be fit for a particular purpose or do not contain any weaknesses, vulnerabilities or bugs which could cause, inter alia, the complete loss of ETH Tokens and/or GET Tokens.

12 RISKS

12.1 You understand and accept the risks in connection with the Exchange as non-exhaustively set forth above and hereinafter. In particular, but not concluding, you understand the following inherent risks:

- *Risk of software weaknesses:* You understand and accept that the Smart Contract System concept, the underlying software application and software platform (i.e. the Ethereum blockchain) is still in an early development stage and unproven, why there is no warranty that the process for creating GET Tokens will be uninterrupted or error-free and why there is an inherent risk that the software could contain weaknesses, vulnerabilities or bugs causing, inter alia, the complete loss of ETH Tokens and/or GET Tokens.
- *Regulatory risk:* You understand and accept that the blockchain technology allows new forms of interaction and that it is possible that certain jurisdictions will apply existing regulations on, or introduce new regulations addressing, blockchain technology based applications, which may be contrary to the current setup of the Smart Contract System and GET-Protocol and which may, *inter alia*, result in substantial modifications of the Smart Contract System and/or the GET-Protocol, including its termination and the loss of your GET Tokens.
- *Risk of abandonment / lack of success:* You understand and accept that the creation of the GET Tokens and the development of the GET-Protocol may be abandoned for a number of reasons, including lack of interest from the public, lack of funding, lack of commercial success or prospects (e.g. caused by competing projects). You therefore understand that there is no assurance that, even if the GET-Protocol is partially or fully developed and launched, you will be able to exchange the GET Tokens with the Foundation or whether you will be able or continue to be able to exchange them with other persons on the open market.
- *Risk of limited use:* You understand and accept that the GET Tokens you obtain during the Exchange will not be allowed to fully interact with the GET-Protocol, and that it is uncertain whether your GET Tokens in the future will be allowed to interact with the GET-Protocol.
- *Risk associated with other applications:* You understand and accept that the GET-Protocol may give rise to other, alternative projects, promoted by unaffiliated third parties, under which GET Tokens will have no intrinsic value.
- *Risk of loss of private key:* GET Tokens can only be accessed by using an Ethereum wallet with a combination of your account information (address), private key and password. The private key is encrypted with a password. You understand and accept that if this private key file or password respectively got lost or stolen, the obtained GET Tokens associated with your account (address) or password will be unrecoverable and will be permanently lost.
- *Risk of theft:* You understand and accept that the Smart Contract System concept, the underlying software application and software platform (i.e. the Ethereum blockchain) may be exposed to

attacks by hackers or other individuals that that could result in theft or loss of GET Tokens or ETH Tokens, impacting the ability to develop the GET-Protocol.

- *Risk of Ethereum mining attacks:* You understand and accept that, as with other cryptocurrencies, the blockchain used for the Smart Contract System is susceptible to mining attacks, including but not limited to double-spend attacks, majority mining power attacks, "selfish-mining" attacks, and race condition attacks. Any successful attacks present a risk to the Smart Contract System, expected proper execution and sequencing of GET Token transactions, and expected proper execution and sequencing of contract computations.
- *Risk of volatility:* You understand that the value of ETH Tokens may appreciate or depreciate materially during the period in which the Exchange is still conditional on the Activation Threshold being met, or if applicable to you, the period during which you can cancel the Exchange, and that you bear the risk for any loss of value or loss of opportunity with respect to the ETH Tokens that are transferred back to you should the Activation Threshold not be met or when you, if applicable to you, cancel the Exchange.
- *Risk of Miner Decisions:* You understand and accept that the network of miners will be ultimately in control of the Smart Contract System. You understand that a majority of these miners could agree at any point to make changes to the official Smart Contract System and to run the new version of the Smart Contract System. Such a scenario could lead to GET Tokens losing intrinsic value.

13 NO LIABILITY

13.1 You acknowledge and agree that, to the fullest extent permitted by any applicable law, you will not hold the Foundation, its directors and its employees, and any auxiliary parties such as subcontractors, auditors, and advisors (*hulppersonen*) liable for any and all damages or injury whatsoever caused by or related to the Exchange, the use of, or the inability to use, GET Tokens under any cause or action whatsoever of any kind in any jurisdiction, including, without limitation, actions for breach of warranty, breach of contract or tort (including negligence) and that aforementioned parties shall not be liable for any indirect, incidental, special, exemplary or consequential damages, including for loss of profits, goodwill or data, in any way whatsoever arising out of the Exchange, the use of, or the inability to use GET Tokens.

13.2 You further specifically acknowledge that, to the fullest extent permitted by any applicable law, the Foundation, its directors and its employees, and any auxiliary parties such as subcontractors, auditors and advisors (*hulppersonen*) are not liable to you, and you agree not to seek to hold them liable, for the conduct of third parties, and that the risk of the Exchange and of holding and/or using GET Tokens rests entirely with you.

13.3 You agree, to the fullest extent permitted by any applicable law, to indemnify and hold harmless (*vrijwaren*) the Foundation, its directors and employees and any auxiliary persons such as subcontractors, auditors and advisors (*hulppersonen*) against any claims from parties to whom you subsequently transfer or with whom you subsequently exchange the GET Tokens you obtained as a result of the Exchange.

13.4 Should you act as an agent or intermediary in performing the Exchange, you agree, to the fullest extent permitted by any applicable law, to indemnify and hold harmless (*vrijwaren*) the Foundation, its directors and employees and any auxiliary persons such as subcontractors, auditors and advisors (*hulppersonen*) against any claims from your principals.

13.5 By performing the Exchange and/or by holding, exchanging or using GET Tokensyou agree, to the fullest extent permitted by any applicable law, not to hold the Foundation, its directors and its employees and any auxiliary parties such as subcontractors, auditors and advisors (*hulppersonen*) liable for any tax liability associated with or arising from the Exchange, ownership, exchange or use of GET Tokens or from any other action or transaction related to the GET-Protocol.

13.6 By creating, holding or using GET Tokens, and to the fullest extent permitted by any applicable law, you agree not to hold the Foundation, its directors and its employees or any auxiliary parties such as subcontractors, auditors and advisors (*hulppersonen*) liable for any regulatory implications or liability

associated with or arising from the Exchange, the ownership, exchange or use of GET Tokens or from any other action or transaction related to the GET-Protocol.

13.7 The Foundation, its directors and employees and any auxiliary persons such as subcontractors, auditors and advisors (*hulppersonen*) are, to the fullest extent permitted by any applicable law, not liable for any damages you incur in relation to the Exchange, the use of, or the inability to use GET Tokens, should any of the representations and warranties set out in chapter 9 be false, incorrect, misleading or incomplete.

13.8 Parties mentioned in this chapter that are not a Party may invoke the content of this chapter against you as third party stipulation (*derdenbeding*).

14 MISCELLANEOUS

14.1 The Foundation can be reached via e-mail address for any questions or complaints.

14.2 Dutch law prescribes that consumers (people performing the Exchange outside their profession or trade) are also informed of the following: (i) the Foundation is not bound by any code of conduct; (ii) Dutch law gives consumers certain safeguards to ensure products/services delivered must be in conformity with the agreement pursuant to which they were delivered.

14.3 In the event any provision of this Agreement is or becomes invalid or unenforceable, the validity of the other provisions shall remain unaffected. In lieu of the invalid or unenforceable provision, a valid and enforceable provision shall be (deemed to be) agreed upon, which as closely as possible corresponds to the intended purpose of the invalid or unenforceable provision. The same shall apply to any supplementary interpretation of any of the terms of this Agreement.

14.4 This Agreement contains the entire agreement between the Parties with respect to the subject matter hereof and supersedes all prior agreements and understandings, public or private statements with respect to the subject matter hereof. There are no collateral oral agreements. Amendments to this Agreement must be made in writing in order to become enforceable and effective, unless mandatory law requires a stricter form.

14.5 The Smart Contract System is deployed in Amsterdam. Consequently, the Exchange and the development of the GET-Protocol are considered to be executed in Amsterdam.

14.6 The applicable law is Dutch law. Any dispute arising out of or in connection with the Exchange or this Agreement shall be finally settled by the court of Amsterdam.

***

Annex 1 – Smart contract system code

```
/**
 * Math operations with safety checks
 */
contract SafeMath {
  function safeMul(uint a, uint b) internal returns (uint) {
    uint c = a * b;
    assert(a == 0 || c / a == b);
    return c;
  }
  function safeDiv(uint a, uint b) internal returns (uint) {
    assert(b > 0);
    uint c = a / b;
    assert(a == b * c + a % b);
    return c;
  }
  function safeSub(uint a, uint b) internal returns (uint) {
    assert(b <= a);
    return a - b;
  }
  function safeAdd(uint a, uint b) internal returns (uint) {
    uint c = a + b;
    assert(c>=a && c>=b);
    return c;
  }
  function max64(uint64 a, uint64 b) internal constant returns (uint64) {
    return a >= b ? a : b;
  }
  function min64(uint64 a, uint64 b) internal constant returns (uint64) {
    return a < b ? a : b;
  }
  function max256(uint256 a, uint256 b) internal constant returns (uint256) {
    return a >= b ? a : b;
  }
```

```
  function min256(uint256 a, uint256 b) internal constant returns (uint256) {
    return a < b ? a : b;
  }
}
/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  uint256 public totalSupply;
  function balanceOf(address who) constant returns (uint256);
  function transfer(address to, uint256 value) returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) constant returns (uint256);
  function transferFrom(address from, address to, uint256 value) returns (bool);
  function approve(address spender, uint256 value) returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
```

```
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
  address public owner;
  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to the sender
   * account.
   */
  function Ownable() {
    owner = msg.sender;
  }
  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }
  /**
   * @dev Allows the current owner to transfer control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function transferOwnership(address newOwner) onlyOwner {
    if (newOwner != address(0)) {
      owner = newOwner;
    }
  }
}
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
```

```
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/*
 * Haltable
 *
 * Abstract contract that allows children to implement an
 * emergency stop mechanism. Differs from Pausable by causing a throw when in halt mode.
 *
 *
 * Originally envisioned in FirstBlood ICO contract.
 */
contract Haltable is Ownable {
  bool public halted;
  modifier stopInEmergency {
    if (halted) throw;
    _;
  }
  modifier stopNonOwnersInEmergency {
    if (halted && msg.sender != owner) throw;
    _;
  }
  modifier onlyInEmergency {
    if (!halted) throw;
    _;
  }
  // called by the owner on emergency, triggers stopped state
  function halt() external onlyOwner {
    halted = true;
  }
```

```
  // called by the owner on end of emergency, returns to normal state
  function unhalt() external onlyOwner onlyInEmergency {
    halted = false;
  }
}
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * Interface for defining crowdsale pricing.
 */
contract PricingStrategy {
  /** Interface declaration. */
  function isPricingStrategy() public constant returns (bool) {
    return true;
  }
  /** Self check if all references are correctly set.
   *
   * Checks that pricing strategy matches crowdsale parameters.
   */
  function isSane(address crowdsale) public constant returns (bool) {
    return true;
  }
  /**
   * @dev Pricing tells if this is a presale purchase or not.
     @param purchaser Address of the purchaser
     @return False by default, true if a presale purchaser
   */
  function isPresalePurchase(address purchaser) public constant returns (bool) {
    return false;
  }
  /**
   * When somebody tries to buy tokens for X eth, calculate how many tokens they get.
```

```
   *
   *
   * @param value - What is the value of the transaction send in as wei
   * @param tokensSold - how much tokens have been sold this far
   * @param weiRaised - how much money has been raised this far in the main token sale - this number excludes presale
   * @param msgSender - who is the investor of this transaction
   * @param decimals - how many decimal units the token has
   * @return Amount of tokens the investor receives
   */
  function calculatePrice(uint value, uint weiRaised, uint tokensSold, address msgSender, uint decimals) public constant returns (uint tokenAmount);
}
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * Finalize agent defines what happens at the end of succeseful crowdsale.
 *
 * - Allocate tokens for founders, bounties and community
 * - Make tokens transferable
 * - etc.
 */
contract FinalizeAgent {
  function isFinalizeAgent() public constant returns(bool) {
    return true;
  }
  /** Return true if we can run finalizeCrowdsale() properly.
   *
   * This is a safety check function that doesn't allow crowdsale to begin
   * unless the finalizer has been set up properly.
   */
  function isSane() public constant returns (bool);
  /** Called once by crowdsale finalize() if the sale was success. */
  function finalizeCrowdsale();
```

```
}

/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */

/**
 * A token that defines fractional units as decimals.
 */
contract FractionalERC20 is ERC20 {
  uint public decimals;
}

/**
 * Abstract base contract for token sales.
 *
 * Handle
 * - start and end dates
 * - accepting investments
 * - minimum funding goal and refund
 * - various statistics during the crowdfund
 * - different pricing strategies
 * - different investment policies (require server side customer id, allow only whitelisted addresses)
 *
 */
contract Crowdsale is Haltable {
  /* Max investment count when we are still allowed to change the multisig address */
  uint public MAX_INVESTMENTS_BEFORE_MULTISIG_CHANGE = 5;
  using SafeMathLib for uint;
  /* The token we are selling */
  FractionalERC20 public token;
  /* How we are going to price our offering */
  PricingStrategy public pricingStrategy;
  /* Post-success callback */
  FinalizeAgent public finalizeAgent;
  /* tokens will be transfered from this address */
```

```
address public multisigWallet;

/* if the funding goal is not reached, investors may withdraw their funds */

uint public minimumFundingGoal;

/* the UNIX timestamp start date of the crowdsale */

uint public startsAt;

/* the UNIX timestamp end date of the crowdsale */

uint public endsAt;

/* the number of tokens already sold through this contract*/

uint public tokensSold = 0;

/* How many wei of funding we have raised */

uint public weiRaised = 0;

/* Calculate incoming funds from presale contracts and addresses */

uint public presaleWeiRaised = 0;

/* How many distinct addresses have invested */

uint public investorCount = 0;

/* How much wei we have returned back to the contract after a failed crowdfund. */

uint public loadedRefund = 0;

/* How much wei we have given back to investors.*/

uint public weiRefunded = 0;

/* Has this crowdsale been finalized */

bool public finalized;

/* Do we need to have unique contributor id for each customer */

bool public requireCustomerId;

/**

  * Do we verify that contributor has been cleared on the server side (accredited investors only).

  * This method was first used in FirstBlood crowdsale to ensure all contributors have accepted terms on sale (on the web).

  */

bool public requiredSignedAddress;

/* Server side address that signed allowed contributors (Ethereum addresses) that can participate the crowdsale */

address public signerAddress;

/** How much ETH each address has invested to this crowdsale */

mapping (address => uint256) public investedAmountOf;

/** How much tokens this crowdsale has credited for each investor address */

mapping (address => uint256) public tokenAmountOf;

/** Addresses that are allowed to invest even before ICO offical opens. For testing, for ICO partners, etc. */
```

```
  mapping (address => bool) public earlyParticipantWhitelist;

  /** This is for manul testing for the interaction from owner wallet. You can set it to any value and inspect this in blockchain explorer to see that crowdsale interaction works. */

  uint public ownerTestValue;

  /** State machine

   *

   * - Preparing: All contract initialization calls and variables have not been set yet

   * - Prefunding: We have not passed start time yet

   * - Funding: Active crowdsale

   * - Success: Minimum funding goal reached

   * - Failure: Minimum funding goal not reached before ending time

   * - Finalized: The finalized has been called and succesfully executed

   * - Refunding: Refunds are loaded on the contract for reclaim.

   */

  enum State{Unknown, Preparing, PreFunding, Funding, Success, Failure, Finalized, Refunding}

  // A new investment was made

  event Invested(address investor, uint weiAmount, uint tokenAmount, uint128 customerId);

  // Refund was processed for a contributor

  event Refund(address investor, uint weiAmount);

  // The rules were changed what kind of investments we accept

  event InvestmentPolicyChanged(bool newRequireCustomerId, bool newRequiredSignedAddress, address newSignerAddress);

  // Address early participation whitelist status changed

  event Whitelisted(address addr, bool status);

  // Crowdsale end time has been changed

  event EndsAtChanged(uint newEndsAt);

  function Crowdsale(address _token, PricingStrategy _pricingStrategy, address _multisigWallet, uint _start, uint _end, uint _minimumFundingGoal) {

    owner = msg.sender;

    token = FractionalERC20(_token);

    setPricingStrategy(_pricingStrategy);

    multisigWallet = _multisigWallet;

    if(multisigWallet == 0) {

        throw;

    }

    if(_start == 0) {

        throw;
```

```
  }
  startsAt = _start;
  if(_end == 0) {
      throw;
  }
  endsAt = _end;
  // Don't mess the dates
  if(startsAt >= endsAt) {
      throw;
  }
  // Minimum funding goal can be zero
  minimumFundingGoal = _minimumFundingGoal;
}
/**
 * Don't expect to just send in money and get tokens.
 */
function() payable {
  throw;
}
/**
 * Make an investment.
 *
 * Crowdsale must be running for one to invest.
 * We must have not pressed the emergency brake.
 *
 * @param receiver The Ethereum address who receives the tokens
 * @param customerId (optional) UUID v4 to track the successful payments on the server side
 *
 */
function investInternal(address receiver, uint128 customerId) stopInEmergency private {
  // Determine if it's a good time to accept investment from this participant
  if(getState() == State.PreFunding) {
    // Are we whitelisted for early deposit
    if(!earlyParticipantWhitelist[receiver]) {
      throw;
```

```
    }
  } else if(getState() == State.Funding) {
    // Retail participants can only come in when the crowdsale is running
    // pass
  } else {
    // Unwanted state
    throw;
  }
  uint weiAmount = msg.value;
  // Account presale sales separately, so that they do not count against pricing tranches
  uint tokenAmount = pricingStrategy.calculatePrice(weiAmount, weiRaised - presaleWeiRaised, tokensSold, msg.sender,
token.decimals());
  if(tokenAmount == 0) {
    // Dust transaction
    throw;
  }
  if(investedAmountOf[receiver] == 0) {
     // A new investor
     investorCount++;
  }
  // Update investor
  investedAmountOf[receiver] = investedAmountOf[receiver].plus(weiAmount);
  tokenAmountOf[receiver] = tokenAmountOf[receiver].plus(tokenAmount);
  // Update totals
  weiRaised = weiRaised.plus(weiAmount);
  tokensSold = tokensSold.plus(tokenAmount);
  if(pricingStrategy.isPresalePurchase(receiver)) {
     presaleWeiRaised = presaleWeiRaised.plus(weiAmount);
  }
  // Check that we did not bust the cap
  if(isBreakingCap(weiAmount, tokenAmount, weiRaised, tokensSold)) {
    throw;
  }
  assignTokens(receiver, tokenAmount);
  // Pocket the money
  if(!multisigWallet.send(weiAmount)) throw;
```

```
  // Tell us invest was success
  Invested(receiver, weiAmount, tokenAmount, customerId);
 }
 /**
  * Preallocate tokens for the early investors.
  *
  * Preallocated tokens have been sold before the actual crowdsale opens.
  * This function mints the tokens and moves the crowdsale needle.
  *
  * Investor count is not handled; it is assumed this goes for multiple investors
  * and the token distribution happens outside the smart contract flow.
  *
  * No money is exchanged, as the crowdsale team already have received the payment.
  *
  * @param fullTokens tokens as full tokens - decimal places added internally
  * @param weiPrice Price of a single full token in wei
  *
  */
 function preallocate(address receiver, uint fullTokens, uint weiPrice) public onlyOwner {
  uint tokenAmount = fullTokens * 10**token.decimals();
  uint weiAmount = weiPrice * fullTokens; // This can be also 0, we give out tokens for free
  weiRaised = weiRaised.plus(weiAmount);
  tokensSold = tokensSold.plus(tokenAmount);
  investedAmountOf[receiver] = investedAmountOf[receiver].plus(weiAmount);
  tokenAmountOf[receiver] = tokenAmountOf[receiver].plus(tokenAmount);
  assignTokens(receiver, tokenAmount);
  // Tell us invest was success
  Invested(receiver, weiAmount, tokenAmount, 0);
 }
 /**
  * Allow anonymous contributions to this crowdsale.
  */
 function investWithSignedAddress(address addr, uint128 customerId, uint8 v, bytes32 r, bytes32 s) public payable {
    bytes32 hash = sha256(addr);
    if (ecrecover(hash, v, r, s) != signerAddress) throw;
```

```
  if(customerId == 0) throw;  // UUIDv4 sanity check
  investInternal(addr, customerId);
}
/**
 * Track who is the customer making the payment so we can send thank you email.
 */
function investWithCustomerId(address addr, uint128 customerId) public payable {
  if(requiredSignedAddress) throw; // Crowdsale allows only server-side signed participants
  if(customerId == 0) throw;  // UUIDv4 sanity check
  investInternal(addr, customerId);
}
/**
 * Allow anonymous contributions to this crowdsale.
 */
function invest(address addr) public payable {
  if(requireCustomerId) throw; // Crowdsale needs to track participants for thank you email
  if(requiredSignedAddress) throw; // Crowdsale allows only server-side signed participants
  investInternal(addr, 0);
}
/**
 * Invest to tokens, recognize the payer and clear his address.
 *
 */
function buyWithSignedAddress(uint128 customerId, uint8 v, bytes32 r, bytes32 s) public payable {
  investWithSignedAddress(msg.sender, customerId, v, r, s);
}
/**
 * Invest to tokens, recognize the payer.
 *
 */
function buyWithCustomerId(uint128 customerId) public payable {
  investWithCustomerId(msg.sender, customerId);
}
/**
 * The basic entry point to participate the crowdsale process.
```

```
 *
 * Pay for funding, get invested tokens back in the sender address.
 */
function buy() public payable {
  invest(msg.sender);
}
/**
 * Finalize a succcesful crowdsale.
 *
 * The owner can triggre a call the contract that provides post-crowdsale actions, like releasing the tokens.
 */
function finalize() public inState(State.Success) onlyOwner stopInEmergency {
  // Already finalized
  if(finalized) {
    throw;
  }
  // Finalizing is optional. We only call it if we are given a finalizing agent.
  if(address(finalizeAgent) != 0) {
    finalizeAgent.finalizeCrowdsale();
  }
  finalized = true;
}
/**
 * Allow to (re)set finalize agent.
 *
 * Design choice: no state restrictions on setting this, so that we can fix fat finger mistakes.
 */
function setFinalizeAgent(FinalizeAgent addr) onlyOwner {
  finalizeAgent = addr;
  // Don't allow setting bad agent
  if(!finalizeAgent.isFinalizeAgent()) {
    throw;
  }
}
/**
```

```
 * Set policy do we need to have server-side customer ids for the investments.
 *
 */
function setRequireCustomerId(bool value) onlyOwner {
  requireCustomerId = value;
  InvestmentPolicyChanged(requireCustomerId, requiredSignedAddress, signerAddress);
}
/**
 * Set policy if all investors must be cleared on the server side first.
 *
 * This is e.g. for the accredited investor clearing.
 *
 */
function setRequireSignedAddress(bool value, address _signerAddress) onlyOwner {
  requiredSignedAddress = value;
  signerAddress = _signerAddress;
  InvestmentPolicyChanged(requireCustomerId, requiredSignedAddress, signerAddress);
}
/**
 * Allow addresses to do early participation.
 *
 * TODO: Fix spelling error in the name
 */
function setEarlyParicipantWhitelist(address addr, bool status) onlyOwner {
  earlyParticipantWhitelist[addr] = status;
  Whitelisted(addr, status);
}
/**
 * Allow crowdsale owner to close early or extend the crowdsale.
 *
 * This is useful e.g. for a manual soft cap implementation:
 * - after X amount is reached determine manual closing
 *
 * This may put the crowdsale to an invalid state,
 * but we trust owners know what they are doing.
```

```
 *
 */
function setEndsAt(uint time) onlyOwner {
  if(now > time) {
    throw; // Don't change past
  }
  endsAt = time;
  EndsAtChanged(endsAt);
}
/**
 * Allow to (re)set pricing strategy.
 *
 * Design choice: no state restrictions on the set, so that we can fix fat finger mistakes.
 */
function setPricingStrategy(PricingStrategy _pricingStrategy) onlyOwner {
  pricingStrategy = _pricingStrategy;
  // Don't allow setting bad agent
  if(!pricingStrategy.isPricingStrategy()) {
    throw;
  }
}
/**
 * Allow to change the team multisig address in the case of emergency.
 *
 * This allows to save a deployed crowdsale wallet in the case the crowdsale has not yet begun
 * (we have done only few test transactions). After the crowdsale is going
 * then multisig address stays locked for the safety reasons.
 */
function setMultisig(address addr) public onlyOwner {
  // Change
  if(investorCount > MAX_INVESTMENTS_BEFORE_MULTISIG_CHANGE) {
    throw;
  }
  multisigWallet = addr;
}
```

```
/**
 * Allow load refunds back on the contract for the refunding.
 *
 * The team can transfer the funds back on the smart contract in the case the minimum goal was not reached..
 */
function loadRefund() public payable inState(State.Failure) {
  if(msg.value == 0) throw;
  loadedRefund = loadedRefund.plus(msg.value);
}
/**
 * Investors can claim refund.
 *
 * Note that any refunds from proxy buyers should be handled separately,
 * and not through this contract.
 */
function refund() public inState(State.Refunding) {
  uint256 weiValue = investedAmountOf[msg.sender];
  if (weiValue == 0) throw;
  investedAmountOf[msg.sender] = 0;
  weiRefunded = weiRefunded.plus(weiValue);
  Refund(msg.sender, weiValue);
  if (!msg.sender.send(weiValue)) throw;
}
/**
 * @return true if the crowdsale has raised enough money to be a successful.
 */
function isMinimumGoalReached() public constant returns (bool reached) {
  return weiRaised >= minimumFundingGoal;
}
/**
 * Check if the contract relationship looks good.
 */
function isFinalizerSane() public constant returns (bool sane) {
  return finalizeAgent.isSane();
}
```

```
/**
 * Check if the contract relationship looks good.
 */
function isPricingSane() public constant returns (bool sane) {
  return pricingStrategy.isSane(address(this));
}
/**
 * Crowdfund state machine management.
 *
 * We make it a function and do not assign the result to a variable, so there is no chance of the variable being stale.
 */
function getState() public constant returns (State) {
  if(finalized) return State.Finalized;
  else if (address(finalizeAgent) == 0) return State.Preparing;
  else if (!finalizeAgent.isSane()) return State.Preparing;
  else if (!pricingStrategy.isSane(address(this))) return State.Preparing;
  else if (block.timestamp < startsAt) return State.PreFunding;
  else if (block.timestamp <= endsAt && !isCrowdsaleFull()) return State.Funding;
  else if (isMinimumGoalReached()) return State.Success;
  else if (!isMinimumGoalReached() && weiRaised > 0 && loadedRefund >= weiRaised) return State.Refunding;
  else return State.Failure;
}
/** This is for manual testing of multisig wallet interaction */
function setOwnerTestValue(uint val) onlyOwner {
  ownerTestValue = val;
}
/** Interface marker. */
function isCrowdsale() public constant returns (bool) {
  return true;
}
//
// Modifiers
//
/** Modified allowing execution only if the crowdsale is currently running.  */
modifier inState(State state) {
```

```
    if(getState() != state) throw;
    _;
  }
  //
  // Abstract functions
  //
  /**
   * Check if the current invested breaks our cap rules.
   *
   *
   * The child contract must define their own cap setting rules.
   * We allow a lot of flexibility through different capping strategies (ETH, token count)
   * Called from invest().
   *
   * @param weiAmount The amount of wei the investor tries to invest in the current transaction
   * @param tokenAmount The amount of tokens we try to give to the investor in the current transaction
   * @param weiRaisedTotal What would be our total raised balance after this transaction
   * @param tokensSoldTotal What would be our total sold tokens count after this transaction
   *
   * @return true if taking this investment would break our cap rules
   */
  function isBreakingCap(uint weiAmount, uint tokenAmount, uint weiRaisedTotal, uint tokensSoldTotal) constant returns (bool limitBroken);
  /**
   * Check if the current crowdsale is full and we can no longer sell any tokens.
   */
  function isCrowdsaleFull() public constant returns (bool);
  /**
   * Create new tokens or transfer issued tokens to the investor depending on the cap model.
   */
  function assignTokens(address receiver, uint tokenAmount) private;
}
/**
 * ICO crowdsale contract that is capped by amout of tokens.
 *
 * - Tokens are dynamically created during the crowdsale
```

```
 *
 *
 */
contract MintedTokenCappedCrowdsale is Crowdsale {
  /* Maximum amount of tokens this crowdsale can sell. */
  uint public maximumSellableTokens;
  function MintedTokenCappedCrowdsale(address _token, PricingStrategy _pricingStrategy, address _multisigWallet, uint _start, uint _end,
uint _minimumFundingGoal, uint _maximumSellableTokens) Crowdsale(_token, _pricingStrategy, _multisigWallet, _start, _end,
_minimumFundingGoal) {
    maximumSellableTokens = _maximumSellableTokens;
  }
  /**
   * Called from invest() to confirm if the curret investment does not break our cap rule.
   */
  function isBreakingCap(uint weiAmount, uint tokenAmount, uint weiRaisedTotal, uint tokensSoldTotal) constant returns (bool
limitBroken) {
    return tokensSoldTotal > maximumSellableTokens;
  }
  function isCrowdsaleFull() public constant returns (bool) {
    return tokensSold >= maximumSellableTokens;
  }
  /**
   * Dynamically create tokens and assign them to the investor.
   */
  function assignTokens(address receiver, uint tokenAmount) private {
    MintableToken mintableToken = MintableToken(token);
    mintableToken.mint(receiver, tokenAmount);
  }
}

/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * Standard ERC20 token with Short Hand Attack and approve() race condition mitigation.
```

```
 *
 * Based on code by FirstBlood:
 * https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
contract StandardToken is ERC20, SafeMath {
  /* Token supply got increased and a new owner received these tokens */
  event Minted(address receiver, uint amount);
  /* Actual balances of token holders */
  mapping(address => uint) balances;
  /* approve() allowances */
  mapping (address => mapping (address => uint)) allowed;
  /* Interface declaration */
  function isToken() public constant returns (bool weAre) {
    return true;
  }
  function transfer(address _to, uint _value) returns (bool success) {
    balances[msg.sender] = safeSub(balances[msg.sender], _value);
    balances[_to] = safeAdd(balances[_to], _value);
    Transfer(msg.sender, _to, _value);
    return true;
  }
  function transferFrom(address _from, address _to, uint _value) returns (bool success) {
    uint _allowance = allowed[_from][msg.sender];
    balances[_to] = safeAdd(balances[_to], _value);
    balances[_from] = safeSub(balances[_from], _value);
    allowed[_from][msg.sender] = safeSub(_allowance, _value);
    Transfer(_from, _to, _value);
    return true;
  }
  function balanceOf(address _owner) constant returns (uint balance) {
    return balances[_owner];
  }
  function approve(address _spender, uint _value) returns (bool success) {
    // To change the approve amount you first have to reduce the addresses`
    //  allowance to zero by calling `approve(_spender, 0)` if it is not
```

```
    //  already 0 to mitigate the race condition described here:
    //  https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
    if ((_value != 0) && (allowed[msg.sender][_spender] != 0)) throw;
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
  }
  function allowance(address _owner, address _spender) constant returns (uint remaining) {
    return allowed[_owner][_spender];
  }
}
/**
 * This smart contract code is Copyright 2017 TokenMarket Ltd. For more information see https://tokenmarket.net
 *
 * Licensed under the Apache License, version 2.0: https://github.com/TokenMarketNet/ico/blob/master/LICENSE.txt
 */
/**
 * Safe unsigned safe math.
 *
 * https://blog.aragon.one/library-driven-development-in-solidity-2bebcaf88736#.750gwtwli
 *
 * Originally from https://raw.githubusercontent.com/AragonOne/zeppelin-solidity/master/contracts/SafeMathLib.sol
 *
 * Maintained here until merged to mainline zeppelin-solidity.
 *
 */
library SafeMathLib {
  function times(uint a, uint b) returns (uint) {
    uint c = a * b;
    assert(a == 0 || c / a == b);
    return c;
  }
  function minus(uint a, uint b) returns (uint) {
    assert(b <= a);
    return a - b;
```

```
  }
  function plus(uint a, uint b) returns (uint) {
    uint c = a + b;
    assert(c>=a);
    return c;
  }
}
/**
 * A token that can increase its supply by another contract.
 *
 * This allows uncapped crowdsale by dynamically increasing the supply when money pours in.
 * Only mint agents, contracts whitelisted by owner, can mint new tokens.
 *
 */
contract MintableToken is StandardToken, Ownable {
  using SafeMathLib for uint;
  bool public mintingFinished = false;
  /** List of agents that are allowed to create new tokens */
  mapping (address => bool) public mintAgents;
  event MintingAgentChanged(address addr, bool state  );
  /**
   * Create new tokens and allocate them to an address..
   *
   * Only callably by a crowdsale contract (mint agent).
   */
  function mint(address receiver, uint amount) onlyMintAgent canMint public {
    totalSupply = totalSupply.plus(amount);
    balances[receiver] = balances[receiver].plus(amount);
    // This will make the mint transaction apper in EtherScan.io
    // We can remove this after there is a standardized minting event
    Transfer(0, receiver, amount);
  }
  /**
   * Owner can allow a crowdsale contract to mint new tokens.
   */
```

```
  function setMintAgent(address addr, bool state) onlyOwner canMint public {
    mintAgents[addr] = state;
    MintingAgentChanged(addr, state);
  }
  modifier onlyMintAgent() {
    // Only crowdsale contracts are allowed to mint new tokens
    if(!mintAgents[msg.sender]) {
        throw;
    }
    _;
  }
  /** Make sure we are not done yet. */
  modifier canMint() {
    if(mintingFinished) throw;
    _;
  }
}
contract GetWhitelist is Ownable {
    using SafeMathLib for uint;
    event NewEntry(address whitelisted);
    event NewBatch();
    event EdittedEntry(address whitelisted, uint tier);
    event WhitelisterChange(address whitelister, bool iswhitelister);
    struct WhitelistInfo {
        uint presaleAmount;
        uint tier1Amount;
        uint tier2Amount;
        uint tier3Amount;
        uint tier4Amount;
        bool isWhitelisted;
    }
    mapping (address => bool) public whitelisters;

    mapping (address => WhitelistInfo) public entries;
    uint presaleCap;
```

```
uint tier1Cap;
uint tier2Cap;
uint tier3Cap;
uint tier4Cap;
modifier onlyWhitelister() {
    require(whitelisters[msg.sender]);
    _;
}
function GetWhitelist(uint _presaleCap, uint _tier1Cap, uint _tier2Cap, uint _tier3Cap, uint _tier4Cap) {
    presaleCap = _presaleCap;
    tier1Cap = _tier1Cap;
    tier2Cap = _tier2Cap;
    tier3Cap = _tier3Cap;
    tier4Cap = _tier4Cap;
}
function isGetWhiteList() constant returns (bool) {
    return true;
}
function acceptBatched(address[] _addresses, bool _isEarly) onlyWhitelister {
    // trying to save up some gas here
    uint _presaleCap;
    if (_isEarly) {
        _presaleCap = presaleCap;
    } else {
        _presaleCap = 0;
    }
    for (uint i=0; i<_addresses.length; i++) {
        entries[_addresses[i]] = WhitelistInfo(
            _presaleCap,
            tier1Cap,
            tier2Cap,
            tier3Cap,
            tier4Cap,
            true
        );
```

```
    }
    NewBatch();
}
function accept(address _address, bool _isEarly) onlyWhitelister {
    require(!entries[_address].isWhitelisted);
    uint _presaleCap;
    if (_isEarly) {
        _presaleCap = presaleCap;
    } else {
        _presaleCap = 0;
    }
    entries[_address] = WhitelistInfo(_presaleCap, tier1Cap, tier2Cap, tier3Cap, tier4Cap, true);
    NewEntry(_address);
}
function subtractAmount(address _address, uint _tier, uint _amount) onlyWhitelister {
    require(_amount > 0);
    require(entries[_address].isWhitelisted);
    if (_tier == 0) {
        entries[_address].presaleAmount = entries[_address].presaleAmount.minus(_amount);
        EdittedEntry(_address, 0);
        return;
    }else if (_tier == 1) {
        entries[_address].tier1Amount = entries[_address].tier1Amount.minus(_amount);
        EdittedEntry(_address, 1);
        return;
    }else if (_tier == 2) {
        entries[_address].tier2Amount = entries[_address].tier2Amount.minus(_amount);
        EdittedEntry(_address, 2);
        return;
    }else if (_tier == 3) {
        entries[_address].tier3Amount = entries[_address].tier3Amount.minus(_amount);
        EdittedEntry(_address, 3);
        return;
    }else if (_tier == 4) {
        entries[_address].tier4Amount = entries[_address].tier4Amount.minus(_amount);
```

```
            EdittedEntry(_address, 4);
            return;
        }
        revert();
    }
    function setWhitelister(address _whitelister, bool _isWhitelister) onlyOwner {
        whitelisters[_whitelister] = _isWhitelister;
        WhitelisterChange(_whitelister, _isWhitelister);
    }
    function setCaps(uint _presaleCap, uint _tier1Cap, uint _tier2Cap, uint _tier3Cap, uint _tier4Cap) onlyOwner {
        presaleCap = _presaleCap;
        tier1Cap = _tier1Cap;
        tier2Cap = _tier2Cap;
        tier3Cap = _tier3Cap;
        tier4Cap = _tier4Cap;
    }
    function() payable {
        revert();
    }
}
contract GetCrowdsale is MintedTokenCappedCrowdsale {
    uint public lockTime;
    FinalizeAgent presaleFinalizeAgent;
    event PresaleUpdated(uint weiAmount, uint tokenAmount);
    function GetCrowdsale(
        uint _lockTime, FinalizeAgent _presaleFinalizeAgent,
        address _token, PricingStrategy _pricingStrategy, address _multisigWallet,
        uint _start, uint _end, uint _minimumFundingGoal, uint _maximumSellableTokens)
        MintedTokenCappedCrowdsale(_token, _pricingStrategy, _multisigWallet,
            _start, _end, _minimumFundingGoal, _maximumSellableTokens)
    {
        require(_presaleFinalizeAgent.isSane());
        require(_lockTime > 0);
        lockTime = _lockTime;
        presaleFinalizeAgent = _presaleFinalizeAgent;
```

```
}
function logPresaleResults(uint tokenAmount, uint weiAmount) returns (bool) {
    require(msg.sender == address(presaleFinalizeAgent));
    weiRaised = weiRaised.plus(weiAmount);
    tokensSold = tokensSold.plus(tokenAmount);
    presaleWeiRaised = presaleWeiRaised.plus(weiAmount);
    PresaleUpdated(weiAmount, tokenAmount);
    return true;
}
// overriden because presaleWeiRaised was not altered and would mess with the TranchePricing
function preallocate(address receiver, uint fullTokens, uint weiPrice) public onlyOwner {
    uint tokenAmount = fullTokens * 10**token.decimals();
    uint weiAmount = weiPrice * fullTokens; // This can be also 0, we give out tokens for free
    weiRaised = weiRaised.plus(weiAmount);
    tokensSold = tokensSold.plus(tokenAmount);
    presaleWeiRaised = presaleWeiRaised.plus(weiAmount);
    investedAmountOf[receiver] = investedAmountOf[receiver].plus(weiAmount);
    tokenAmountOf[receiver] = tokenAmountOf[receiver].plus(tokenAmount);
    assignTokens(receiver, tokenAmount);
    // Tell us invest was success
    Invested(receiver, weiAmount, tokenAmount, 0);
}
function setEarlyParicipantWhitelist(address addr, bool status) onlyOwner {
    // We don't need this function, we have external whitelist
    revert();
}
// added this here because it was not visible by preallocate
function assignTokens(address receiver, uint tokenAmount) private {
    MintableToken mintableToken = MintableToken(token);
    mintableToken.mint(receiver, tokenAmount);
}
function finalize() public inState(State.Success) onlyOwner stopInEmergency {
    require(now > endsAt + lockTime);
    super.finalize();
}
```

```
    function() payable {
        invest(msg.sender);
    }
}
```