Cover

U42 Token
White Paper

Version 1.0
April 23, 2018

Table of Contents

Table of Contents

Table of Contents

Value Proposition

The entertainment industry is fractured. In its analog state, it works in favor of large production houses who are simply masking the reality that they are becoming obsolete. To reach audiences, artists expend more effort jumping through the hoops these houses set up than they do creating content. More often than not, their content is never heard or seen because of the red tape holding them back.

You42 seeks to eliminate this problem by removing the middleman from the equation; no longer is a studio or label needed to bridge the gap between an artist and the audience. You42 is a revolutionary social entertainment platform putting artists and fans in control like never before, enabling new levels of discoverability and success. The platform serves as a venue for distribution, content creation, collaboration, and consumption. You42 brings content creators and consumers together, giving creators a new avenue for monetization, and providing rewards and incentives for users. Users can interact through social, music, video, gaming, and more, bringing together a variety of consumers.

U42 Token

The Token Revolutionizing Entertainment

Blockchain technology affords You42 a method of decentralized control which is core to our vision. By implementing a token economy, You42 will start to decrease the need for third-party merchant services and puts creators in the driver's seat.

The U42 token serves as the economic layer for You42's marketplaces to commercially incentivize participation, discovery and curation. Initially, U42 tokens will be activated in the network by content creators and brands in order to promote content.

Ready Product

- Tokens distributed at the close of the ICO, usable immediately
- Established company background
- 10 years of business history
- Future-proof token economy

Token Use Cases

Using U42 tokens, there are a number of different ways for users, creators and brands to interact with the platform, such as:

- A film director is launching his new feature, “Dead by Midnight,” and wants to drive views of the film across You42.com. Using our integrated advertising network and pre-purchased U42 Tokens, the director can purchase targeted advertising placements directed at his core demographic on the You42 platform.

- A user who sees “Dead by Midnight” and thinks this is one of the top 10 horror films of all time can use U42 Tokens to purchase our in-app currency and “tip” the director for producing such a terrifying experience.

- A top makeup brand that specializes in special effects and Halloween accessories wants to target viewers of “Dead by Midnight” and selects the film to advertise their brand - again purchasing advertising placements using the U42 Token.

Company History

Founded in 2006 under the name Kiz Toys, You42's initial mission was to develop original digital content that accompanied toys, video games and animation properties based on original, cohesive storylines and IP.

With the advent of free-to-play mobile games, we changed our approach to focus solely on the creation and distribution of interactive entertainment with mobile games. In 2014, You42's founders conceived an entertainment product that would bring together games, music, video, news and more, all contained within a shared social space.

In the process of developing this platform, we recognized that we were building an ideal environment for the use of an encrypted token. In 2017, we began planning the campaign for our initial coin offering. Working with legal and accounting advisors, the U42 Tokens will be sold to the public in 2018. Funds collected through the ICO will be used to promote, launch and maintain the You42 platform.

Product Overview

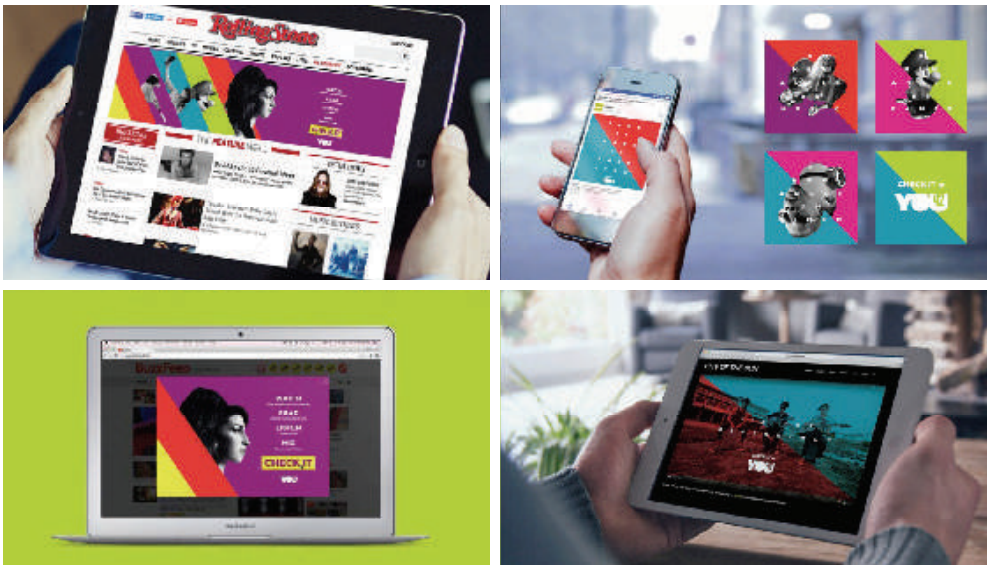
You42 is a social, lifestyle and entertainment platform, focused on the distribution of multiple verticals of entertainment and open access content. Unlike other social platforms, You42 allows users to curate their experiences based on their interests and rewards them for listening to, watching, commenting on and/or sharing content with both tokens and experience points.

While You42 creates a better experience for content consumers, we also help develop the relationship creators have with their audiences. Artists can leverage their digital content to boost monetization, discovery and engagement with fans. By giving content creators a tool to reach their fans across multiple forms of mixed media, we believe this will result in greater loyalty and increased creator revenues.

However, improved engagement isn't the only way we help content creators. Discovery and distribution of content to new consumers is equally important. You42 provides an avenue for content creators to reach new consumers directly. Our platform boasts discoverability at its core by bringing consumers from different media—such as gaming and music—and allows them to experience content in new collaborative and creative ways. As an example, fans of a You42 trending music artist may cross over into the audience from a game also on You42 when they learn the artist contributed to the game's soundtrack.

Since one of our biggest goals is to foster better connections between content creators and their audiences, we want You42 to be accessible to users, no matter where they are or what content they love to consume. That's why building You42.com in HTML5 is central to our philosophy, ensuring that it will work seamlessly across all desktop, tablet and mobile browsers, providing users with the best possible experience based on their device type.

Commerce



You42 will offer content owners and creators dynamic commerce capabilities, allowing them to upload, distribute, stream and sell their content via their profile/pages. Essentially, users will be given the tools to create a personalized marketplace for their digital content. You42's commerce functionality will enable sellers to introduce new products, services and brands while engaging with and rewarding fans for their interaction.

Partnerships

You42 has partnered with the following content providers:

Earbits
earbits.com
Earbits was designed with one goal in mind: to make it simple for artists and music lovers to find each other and create meaningful connections. Earbits will stream music built by musicians and music lovers for musicians and music lovers.

Earbits is an online, commercial-free, independent music-streaming service. The Earbits service does not ingest any fees for the consumption of content, and all content is submitted by the owner with the intent of promotion.

MediaNet
mndigital.com
MediaNet has been working for over a decade with the world's leading major labels and thousands of independent labels to provide access to some of the most extensive music catalogs covering multiple content distribution territories. MediaNet will provide a portion of the fulfillment for the You42 Radio module/offering.

Digital Game Publishers
We have partnered with a number of video game publishers to (initially) bring their mobile games (iOS and Android) to the platform along with several HTML5 games via browser publishers.

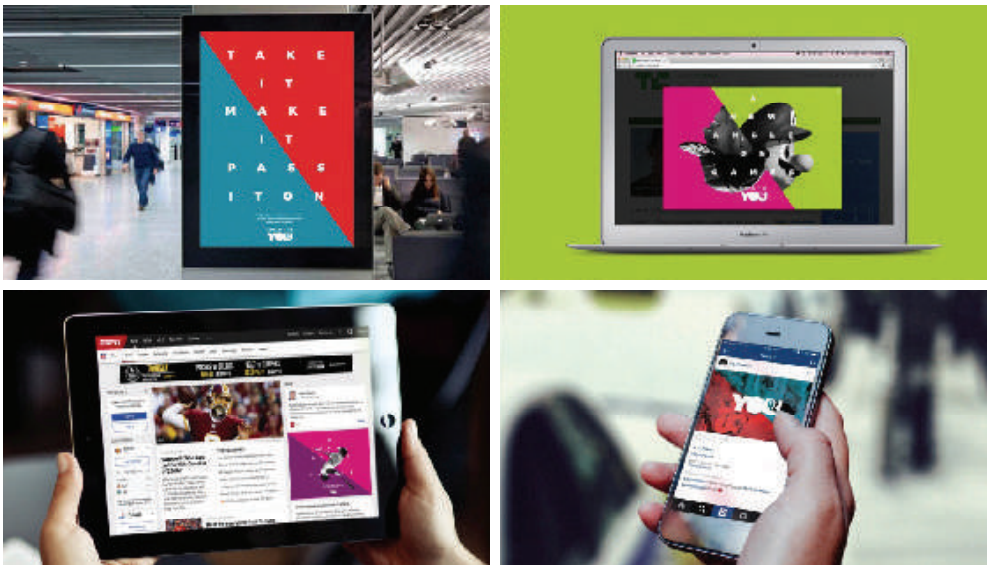
We intend to also monetize the HTML5 games via our own API payment layer and deliver Android games via our own store. This will then be extended to digital PC games.

Product Launch

You42 will launch content across video, music and games; offering a multitude in diverse engagement opportunities.

Content creators can use You42's social hub to reward their fans, giving them exclusive or early access to new and original content. Creators can share photos, videos, live feeds and messages directly to their fan base and monetize/commercialize them through macro- and micro-transactions, advertising and social engagement. Users can engage in a 360-degree, multi-vertical media consumption experience which rewards them for their activities within the platform.

Marketing



You42.com is launching in 2018. Our preliminary marketing push will be through our existing database of beta signups and our community of gamers. As we progress, we will be opening our beta program across the United States, running local and national campaigns such as the following:

- Localized marketing around key cities and metropolitan areas. The marketing campaigns will begin in Atlanta and then move out to New York, San Francisco, Los Angeles, Seattle, etc. We will have on-the-ground street teams running local ad campaigns while also organizing mini-festivals and gigs.

- A comprehensive PR push through mainstream and dedicated entertainment and technology press and influencers.

Service Architecture

Core Technologies

The You42 application runs on a server operating on a LEMP stack.

The core of You42 is built using the Laravel framework. In addition to being robust enough to handle such a large-scale application, this framework offers a number of services that we have integrated to suit the application's needs

Delivery to End Users

As a web-based application, You42 is delivered to end users via web browsers.

Client Player Technologies

The You42 web application allows logged-in users to play audio and video media in their browsers.

U42 Token Overview

Token Launch Summary

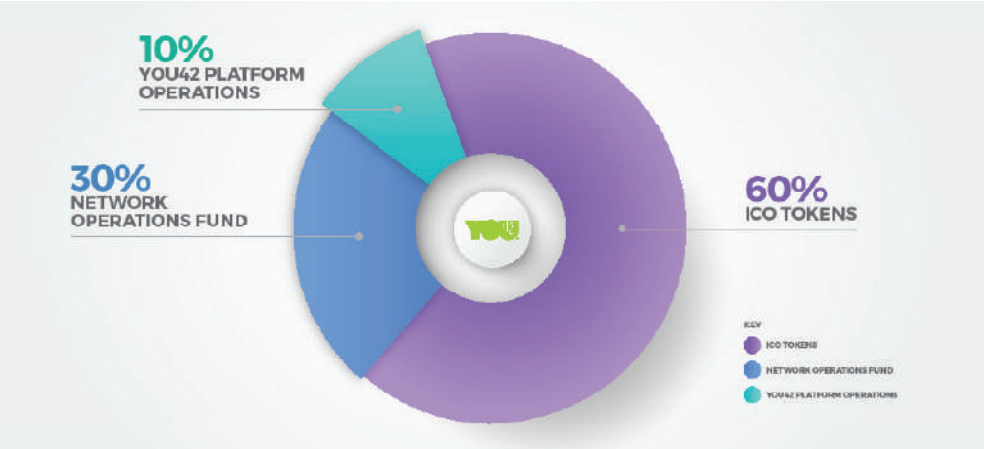
Numbers are subject to change with ETH/USD exchange rates and volatility, but the following are best effort estimates as of April 18, 2018:

- Exchange rate
  - 1 ETH = 1,031 U42 (1U42= 0.0009699321048 ETH)
- Presale to accredited investors: 04.23.18
- Public sale: 07.02.18

Token Distribution

- Lifetime Tokens Available: 525 million (no new U42 Tokens will ever be created)
- ICO: 315 million U42 Tokens (60% of total supply)
- Network Operations Fund: 157.5 million U42 Tokens (30% of total supply)
- You42 Platform Operations & Maintenance: 52.5 million U42 Tokens (10% of total supply)

U42 Token Overview



ICO Budget Allocation (315M Tokens)
All funds will be used to fully launch and sustain the You42 platform.

- **Market Expansion 79.5%**
  Funds will be used to support an internal team of developers that launch, maintain and scale the You42 platform, license professional and semiprofessional digital entertainment content (music, video, games, etc.), and support operational activities that will contribute to the platform's mission. Further, resources allocated to marketing/PR of the platform will focus on continuing to build the You42 brand and executing a competitive user acquisition campaign to grow the network and user base.

- **Administration/Operations 20.5%**
  Legal, security, accounting, and other administrative and operational costs associated with the launch of the U42 Token and You42 platform. Some of these costs include debts accrued during the development of the You42 platform prior to the launch of the U42 Token.

Network Operations Fund (157.5M Tokens)
Perhaps one of the most important token funds, this reserve allows You42 to incentivize and attract both creator talent and advertising partners to the initial stage of the platform. Primarily, this fund will be used to run accelerator programs for highly desirable content creators to provide media for the platform. This, in turn, will encourage users and advertisers to engage with the You42 platform.

You42 Platform Operations (52.5M Tokens)
These tokens will be retained by You42 and dispersed as needed to support the functionality of the platform, engage new partners, and provide quality and unique content to You42 users on an ongoing basis. Further, a portion of these tokens will be allocated to founders, advisors, and employees.

U42 Token Overview

Supply & Logic

The total U42 Token supply is based on reasonable and marketable limits for the ICO as well as a sustainable utility token-driven economy that meets anticipated platform demand.

Total supply = 525,000,000 U42 Tokens

- **Public Sale 1** 15% Bonus
- **Public Sale 2** 10% Bonus
- **Public Sale 3** 5% Bonus
- **Public Sale 4** 0% Bonus

ICO Presale

The ICO pre-sale structure is to accredited investors only in advance of the public sale.

ICO Public Sale

To reward early participants in the You42 network, the public sale phase utilizes bonus purchase percentages based on ICO targets for raising USD through supported cryptocurrencies. As a target is reached, the next public sale phase with a lower bonus percentage is initiated.

ICO Token Burn

There will be a token burn after the public sale phase of the ICO closes. There are a total of 315,000,000 tokens allocated to the ICO; anything 'Not Sold' after the close of the public sale will be burned from the public sale allocation.

*All detailed numbers included in the whitepaper draft are subject to change.

Disclaimer

The purpose of this White Paper is to present You42, a social, lifestyle and entertainment platform, to potential community members who join the You42 community in connection with the proposed U42 Token launch and crowdsale.

The information set forth in this White Paper is not exhaustive and does not imply any elements of a contractual relationship. Its sole purpose is to provide relevant and reasonable information to potential token holders in order for them to determine whether to undertake a thorough analysis of You42 with the intent of acquiring U42 Tokens. Any agreement(s) between You42 or any of its subsidiaries and you to purchase U42 Tokens are to be governed by a separate token sale document. In the event of any inconsistency between this White Paper and the token sale document, the token sale document shall prevail.

Nothing in this White Paper shall be deemed to constitute a prospectus of any sort of a solicitation for investment, nor does it, in any way, pertain to an offering or a solicitation of an offer to buy any securities in any jurisdiction. The White Paper is not composed in accordance with, and is not subject to, laws or regulations of any jurisdiction that are designed to protect investors.

The information, statements, estimates, projections and opinions in this White Paper regarding the projected terms and performance of the You42 platform, are selective and subject to updating, expansion, revision, independent verification and amendment; provided, however, that none of You42 or any of its affiliates hereby undertakes any obligation to provide updates to the information contained herein. As such, neither You42 nor any of its affiliates is making any representation or warranty or undertaking, including those in relation to the truth, accuracy and completeness of any of the information set out in this White Paper.

None of the contents of this White Paper constitutes legal, financial, tax or other advice and we encourage you to consult with the relevant professional advisors independently.

The regulatory status of cryptographic tokens, including any cryptocurrency, digital assets and blockchain technology is unclear or unsettled in many jurisdictions. The publication and dissemination of this White Paper do not imply that any relevant laws, regulations and rules have been complied with. No regulatory authority has examined or approved this White Paper. Where any relevant governmental authority makes changes to existing laws, regulations and/or rules, it may have a material adverse effect and/or impair the ability of the You42 platform to function as intended, or at all.

Disclaimer

This White Paper is for general information purposes only and is not an advertisement. Distribution of this White Paper may be restricted or prohibited by law or regulatory authority in your jurisdiction. Recipients should inform themselves of and comply with all such restrictions or prohibitions and neither You42 nor any of its affiliates accept any liability to any person in relation thereto.

Jurisdiction & Participation Restrictions

The U42 Tokens shall not be sold to any person that is, or is purchasing on behalf of, a citizen or resident of, or a person located in or with a primary residence or domicile in, the People's Republic of China, the Republic of Korea or any other country or jurisdiction in which access to or use of cryptocurrencies and/or digital tokens is prohibited by law, decree, regulation, treaty or administrative act.

You42 may take all necessary and appropriate actions, in its sole discretion, including referral of information to the appropriate authorities, to invalidate any purchases of U42 Tokens in violation of these restrictions.

Section

U42 Token Technical White Paper

Version 1.0
April 23, 2018

Abstract

The U42 Token is a decentralized utility token intended to provide a foundation for content creators, platforms and consumers to build on for a variety of interactions. The U42 Token uses a services model implemented as a smart contract that extends the ERC-20 Ethereum token standard.

This document describes the features of the U42 Token and includes the full specification of the token as an appendix. Potential and actual uses of the token are explored, features of the specification are described and deployment and development information is presented for application implementers.

The examples of use found in this document describe features included in the You42 platform as well as potential uses by other application implementers working with the U42 Token.

In addition to providing a foundation for content platforms and applications, the U42 Token offers a variety of benefits to application implementers. The opt-in structure and service listing approach of the token allow for it to be integrated in a variety of platform and application types. Provisioning and linked transfer features allow for deep integration in fully decentralized applications. The smart contract compatibility offered by the token, particularly in relation to the delegated security and token receipt address mechanisms, allow for the token to be used and extended in many different scenarios.

Summary of Use

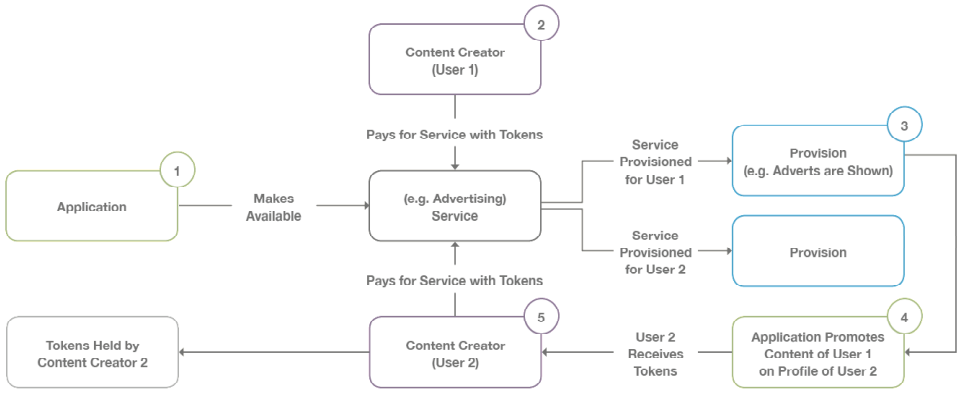


Diagram 1: Simple usage example for the U42 Token

In this example an Application makes an advertising service available to users who can place advertisements on the profile pages of other content-creating users. Users who display advertisements are paid in tokens, which can be used to purchase services.

1. Application makes a service available
2. A content-creating user of that application pays for that advertising service with U42 Tokens
3. The service is provisioned and advertisements are shown
4. The application tracks which users displayed advertisements for that provisioned service and remunerates them with U42 Tokens
5. A different content-creating user (user 2) receives tokens and can use them to purchase their own services

See section "Examples of Use" for expanded versions of the above diagram incorporating other features of the token.

Intended Purpose & Core Design Factors

The U42 Token is intended to provide functionality for both application implementers and content creator users, where the content creator has a variable role in the provisioned application services – e.g., where they can be both service consumer (when purchasing advertisements to promote their content) and service beneficiary (where they benefit from showing advertisements for other content-creating users and services). Content creators can be individual artists and directors, groups of artists, brands and other collaborative, individual or commercial entities that publish creative content.

Furthermore, the U42 Token is designed to meet the requirements of application implementers that offer variable and dynamic services, where the availability and dynamic nature of the services is a factor in how (and when) content creators use those services.

The U42 Token has been designed to provide a core means by which multiple platforms, application types, content creators, brands and other users can interact to provide, consume and benefit from services based on their respective needs at any time.

In order to provide the general purpose utility required by applications, platforms and content creators, the U42 Token has been created with a flexible structure around a standards-compatible (Ethereum ERC-20) core.

Token Technology & Features

The token specifications in Appendix A provide a full description of a set of smart contract methods to provide the features of the U42 Token using the Solidity (^0.4) smart contract language, intended to be deployed on the Ethereum main network (see sections "Token Application Development" and "Testing Applications with the U42 Token" for details of test network facilities).

Use of ERC-20

The U42 Token is an Ethereum ERC-20-compatible smart contract made available on both the Ethereum main network (for production applications) and a test network for the purposes of application testing and development.

The standard ERC-20 methods have been implemented as part of the U42 Token specification (see section "Token Specification") and offer compatibility with other ERC-20-compatible tools.

By using a standard smart contract interface on the leading smart contract platform (Ethereum), the U42 Token is both versatile and extensible. As detailed in this document, it's possible for application implementers and content creators to extend the utility of the U42 Token by interfacing using both standard ERC-20-compatible wallets and other smart contracts deployed to the Ethereum network.

Users (including application operators, content creators and other application users) will need to use an ERC-20 compatible wallet to hold their U42 Tokens. Users should not use exchange-hosted wallets to receive U42 Tokens.

Due to the nature of the ERC-20 token standard, method calls on the smart contract need to be paid (the "gas" required to execute the method needs to be paid) for in Ether. It is expected that all users will maintain a balance of Ether for this purpose. ERC-865 may provide an alternative to this at a future time.

Opt-In Structure

The U42 Token provides a set of methods for interacting with the token. It is not required that an application implementer uses all of these methods, or that they use them in a specific manner. The examples section of this document provides a set of typical use cases, but more combinations are possible. Common groupings of uses would include:

- Applications that make services available for a U42 Token fee and credit users of the application with a portion of those tokens
- Applications that provide dynamic services with variable rates, based on updated and published token/credit rates
- Applications that provide an interface between service users (e.g., advertisers) and service providers (e.g., content creators that will show advertisements) where there is no requirement for a direct relationship between user and provider
- Content creators that purchase services to promote their content and receive payment in tokens for promoting the content of other users
- Content creators that do not use U42 Token-based services but do provide a content platform for other users
- Users that are not content creators but wish to participate in the use of commercial services (e.g., brand advertisers) or consumer services (e.g., the consumption of content)

The examples above are provided by a mix of different methods of the token, as described in full in the token specifications in Appendix A. The rest of this section describes that core token functionality.

Applications

Applications act as a conduit for U42 Tokens by making services available to token holders and dispersing tokens to users who provide utility to the application. Examples of this include:

- Platforms for the publication, delivery and consumption of entertainment media (content)
- Services that integrate with other platforms indirectly (e.g., advertising services that are built with U42 Tokens but displayed on generic platforms)
- Services that are provided to or between content creators (e.g., services used as part of the content creation process)

An application exists by virtue of an application address – an Ethereum address – from which the application operator can make available services and update services based on availability. They can also optionally choose to report information back to the underlying blockchain about service use and payments of tokens to users.

A single commercial entity or platform can operate many different application addresses and may choose to do so as a means to group services or for operational management.

It is expected that applications that provide services that are in part served by content-creating users will maintain a list of wallet addresses for those users, so such that when a content creator or other user is due to receive compensation for its contributions to a service (e.g., by displaying advertisements alongside their content), they can receive payment in an automated fashion.

Services

A service is made available by an application though the listService method of the token smart contract. It includes information about the service (a description), details about how that service can be updated and limits on its consumption (e.g., by specifying a credit rate – see below – and a maximum purchasable volume).

Consumption of services by users is built as a layer on top of the underlying ERC-20 transfer method. When required, a user sends tokens to the transferToService method indicating which service they wish to consume. This process creates a provision (see section "Provisions") for that service and would be reflected in the application in a way that was associated with its description (e.g., a service described as "display advertising: similar artists" would provide an advertising service available to the user inside of that application).

Services can be of different types with different internal mechanics. An application may choose to independently publish information about the different types of services that it provides. Application implementers may choose to use different application identifiers (application addresses) for the purpose of grouping different types of services.

The association of service consumption (e.g., through provision) with applications is up to the application implementer, but could be via monitoring the transferToService method (via its events) or by looking for a specific application reference (see section "Application References").

When a new service is listed by an application a NewService event is created for that application address. This can be monitored by other applications, e.g., for the purpose of service discovery or the automated publishing of information about services.

Service information can be updated by the application, or by a designated set of "Update Addresses" (see section "Delegated Security Model"), e.g., to change the description of a service, its cost in tokens or the maximum amount of that service available for purchase.

An application can choose to specify an alternate receipt address (the default is the application address itself), to which U42 Tokens will be sent when users purchase services via transferToService. In addition to providing increased operational security (see section "Delegated Security Model") this allows application implementers to extend the U42 Token smart contract with their own smart contract functionality, e.g., by publishing a smart contract to the service receipt address. See section "Extending Receipt Addresses" for more information.

Services are created using a numeric identifier that is unique to the application that listed it. Service identifiers are shared across regular and simple services (see section "Simple Services"). The identifiers of services that are removed (no longer available for use) can not be re-used.

Credits

Credits are an abstract concept that can be used by application services to create a total service value based on a known quantity. In addition to providing a volume-based service request, credits can be used by applications to indicate partial or full completion of a service (via updating a provision).

The meaning of a credit is specific to each service, and applications are expected to only use credits for services that have a volume-based provision requirement.

Credits have no special meaning within the stored data of the smart contract. Updating a provision with a remaining credit amount does not check previous credit update values.

Simple Services

The U42 Token provides a mechanism for applications to publish services that do not use the Credits concept and do not require completion at a later stage – i.e., services that are considered to be consumed on use. The listSimpleService and transferToSimpleService are lightweight alternatives to the listService and transferToService methods, designed to provide a cost-effective (e.g., in gas terms) interface for the listing and, most importantly, consumption of services.

Note that simple services share service identifiers with regular services for the same application.

Simple services do not create a provision identifier and immediately log a CompleteSimpleProvision along with the Transfer event.

Applications can choose to implement simple-like services – i.e., those that do not use credits and don't report progress or completion – by using a regular service and setting the maximum permitted credit amount of the service to 1. Creating a simple service in this manner will result in a slightly less cost-effective method call when users come to use the service via transferForService.

Obtaining Listed Service Information

Interested parties (i.e., users and other applications) can obtain information about the services of an application from the getServicesForApplication method. In addition, the getServiceInformation method provides a full set of service information for a specific service. The U42 Token smart contract itself does not provide any hierarchy or structure to the available services of an application, though an application could choose to embed such information in the service description field of the listed service.

As the listService and listSimpleService methods publish NewService and NewSimpleService events it is also possible to monitor the network for newly published services.

Services that have been removed from an application can be discovered using the getRemovedServicesForApplication method or by monitoring the ServiceRemoved event.

The U42 Token smart contract does not provide a direct means to identify new applications, though the NewService event could be monitored for new application addresses (the first time an address lists a service is effectively the creation of an application).

Provisions

When a service (other than a simple service) is acquired by a user, the U42 Token smart contract creates an associated Provision. This provision can be used to further update the smart contract with service progress and completion.

The use and update of provisions is optional, but all calls to the transferToService method return a provision identifier should it be required. Provision identifiers are unique to a particular service.

Provision Updates and Completion

For applications that wish to provide a public update of progress, an updateProvision method is provided by the U42 Token smart contract.

Note that the amount of credits remaining in the provision (as supplied to the updateProvision method) is not validated by the U42 Token smart contract, though applications can use the getProvisionCreditsRemaining method to query the last value that was passed to the updateProvision method.

Provision updates create UpdateProvision events, so can be used to publicly track the status of a provisioned service.

Completed provisions (e.g., an instance of a service that has been completed/provided by the application) can be indicated as such with the completeProvision method on the U42 Token smart contract. This has similar properties to updating a provision, in that it allows for the public tracking of the status of a provisioned service.

The updating and completion of provisions is optional, and the provision identifier provided by acquiring a service can be ignored if not required.

Automated Refunds

The completeProvision method on the U42 Token smart contract also allows for the automation of refunds for services that are multi part (e.g., greater than 1 credit in value). If the completeProvision method is called with a value of credits remaining that is not 0, the smart contract will automatically refund the correct proportion of tokens back to the original acquiring user.

Note that refunds may also be implemented directly in an application using the underlying ERC-20 transfer method instead of updating the completion of a provision.

For applications that do use the provision update and completion mechanisms, it is recommended that the built-in refund mechanism is used as it will prove more cost effective than initiating a separate transfer.

Linked Transfers

Similar to how applications can choose to update the progress and completion of a provisioned service, it is also possible for an application to make transfers in tokens with a specific link to a provisioned service. This will be useful where an application provides a proxy between separate users, e.g., content creators placing advertisements on one another's content.

The transferBecauseOf method is a simple wrapper around the underlying transfer mechanism that additionally logs the application, service and provision identifiers to the TransferBecauseOf event.

The linked transferBecauseOf and transferBecauseOfAggregate methods verify the information passed in about the provision, service and application. If a specified provision does not link to the specified service or the service to the application the method will fail and no tokens will be transferred. This acts as an additional verification step in the sending of tokens as a result of service provision.

In situations where a user is credited in tokens as a result of many provisions of the same service (e.g., a content creator who shows advertisements alongside its content for many other content creators) it may not be cost effective or desired to call the transferBecauseOf method many times. For this reason, a transferBecauseOfAggregate method is provided by the U42 Token smart contract. This aggregate method allows an application to pass a list of provision identifiers and token amounts to the method. This aggregate method only supports a list of transfers for the same service identifier.

Note that there is no built-in mechanism for linking the addresses used in a transfer with a particular user or entity, other than by linking the same address used in other token interactions. Applications that choose to use linked transfers would need to consider how to present this mechanism to end users. Though a normal transfer of tokens (using the underlying ERC-20 transfer method) creates a public record of that address, it doesn't explicitly link it to the provision of a particular service.

Application References

The transferToService and transferToSimpleService methods of the U42 Token smart contract allow for an optional “applicationReference” parameter to be passed when calling methods of the smart contract that transfer tokens. This is intended as an optional reference field for application implementers, and an alternative means of referencing provisioned services with (application) users over reconciling addresses.

The application reference parameter is a numeric identifier that can be passed by a caller of a transfer__ method (typically a content-creating user) when transferring tokens to provision a service. It is not verified by the smart contract and does not need to be unique. The expected contents of the application reference parameter should be communicated to a user who wishes to acquire a service.

It is recommended that applications generate unique, one-time-use references for each service and that they are provided to users within the application itself (or for example via the API of that application to other application users) as a means to associate service provision without publishing user-identifiable information to the smart contract and underlying blockchain.

The StartProvision and CompleteSimpleService events log the contents of the applicationReference parameter, but not as an indexed field (an application would have to monitor all StartProvision and CompleteSimpleService events to inspect the applicationReference parameter).

Delegated Security Model

The U42 Token provides 1.) a split security model with role-specific addresses for creating application services, updating their details and receiving token funds as a result of service consumption, and 2.) an opportunity to extend this by integrating application-specific smart contracts with additional operational security built in.

Role-specific Addresses

The core of the split security model is as follows:

- An application is an address that calls the listService or listSimpleService methods on the U42 Token smart contract. That address becomes the effective administrator for any created services

- When creating a service, an application can specify a list of "Update" addresses that have the ability to modify the details of the service (e.g., its description and cost). Only the application address (effective administrator) can change the authorized update addresses

- When creating a service, an application can specify a receipt address – if specified this address will be used when transferring funds inside of transferToService and transferToSimpleService methods. The application address continues to be the effective administrator and can change the receipt address at any time (including back to itself)

Note that where a receipt address is used for the purpose of token funds security it is strongly recommended that the update addresses feature is also used – a receipt address can be changed by the application address, so any expected lower privileged updates should be done via the update address mechanism.

It is expected that an application that updates services on a regular basis (e.g., for pricing changes) would delegate one or more update addresses such that a compromise of a system used to update the service doesn't also have potential access to received funds/tokens.

The receipt address mechanism can also be used to automate actions on the receipt of funds by means of an application-specific smart contract. See section "Extending Receipt Addresses".

Extending the Security Model

A single application (application address) can create many services. Those services all share the same application address and therefore the same effective administrator. An application owner / controlling entity might choose to create services from several application addresses in order to provide an organization-specific grouping of administrative functions.

In addition to the built-in and proposed multi-application solutions above, application implementers could also choose to implement a smart contract as the interface to application service (application) creation. If the call to the listService or listSimpleService methods comes from a smart contract, that smart contract becomes the effective administrator of the application and its services. That application-specific smart contract can then implement bespoke measures as appropriate for the application or its operational requirements.

Extending Funds and Transfer Receipt with Smart Contracts

The U42 Token is implemented as an Ethereum smart contract – as such it can interface with other smart contracts. In addition to extending the security model for specific application requirements (see section "Extending the Security Model"), the receipt of funds by applications or users could be extended by:

- Deploying a smart contract and specifying it as the receipt address of an application service

- Deploying a smart contract and using its address as the receiver address for content-creating (or other) users

Note that deploying a smart contract for the purpose of receiving token funds should be designed and managed properly. The underlying ERC-20 standard manages a ledger of token-holding addresses, so any smart contract that receives tokens must implement methods to at least be able to withdraw those tokens.

Extending Receipt Addresses

A receipt address may be extended (via smart contract) to provide many purposes, examples might include:

- Locking up tokens in a multi-signature wallet
- Managing “hot” and “cold” token storage (e.g., ensuring a fixed-size hot wallet and separate cold storage mechanism)
- Making payments to service providers (for the underlying application functionality)
- Integrating directly with other application functionality (i.e., a fully integrated distributed application could deploy a smart contract to receive funds and also to make disbursements to content creators via transferBecauseOf)

In order to use this functionality an application must specify on creation (or later change via changeServiceReceiptAddress) a receiptAddress for the service. So long as this address is the address of the smart contract, no further changes are required during the listService or listSimpleService method calls. Note that there is no means for the U42 Token smart contract to verify that the destination address is a smart contract that implements an appropriate withdraw function. It is strongly recommended that such smart contracts are tested via the test version of the U42 Token – see section “Testing Applications with the U42 Token”.

Note that similar functionality could be achieved by using a single deployment smart contract that also received funds (i.e., not using the receipt address) and managed their transfer via some other (presumably internal to that smart contract) method.

Extending User Transfers

Applications send tokens to users as required by their services and functionality – this can be with the underlying ERC-20 transfer method or with the transferBecauseOf and transferBecauseOfAggregate methods of the U42 Token smart contract. All of these transfer methods expect an ERC-20-compatible Ethereum address, which could be a standard user wallet or another smart contract.

A user (e.g., a content creator) who wanted to integrate a more sophisticated payment flow (beyond "send all funds to this address") could implement one by deploying a smart contact with the required functionality and specifying the address of that contract as their user wallet address in a U42 Token application. For example, a content-creating user in the You42 platform might represent a group of artists with different contributions to a piece of content (first X goes to Y, then split Z% between a list of parties). A mechanic like this is an obvious use case for a smart contract, and such a smart contract could be deployed by a user interacting with a U42 Token application without requiring additional features or support from the application implementer or the U42 Token smart contract.

Furthermore, this facility allows a U42 Token user (or users) to implement payment flows and financial distributions across multiple applications and services from different providers. In the case of content creation, a multi-party (e.g., multiple content creators working together on the basis of some commercial agreement) user could conceive of a smart contract that requires all parties to agree (and sign the smart contract) on the basis that it provides an automated distribution of funds based on their commercial agreement.

Use of Logged Events

The U42 Token smart contract uses Solidity events to make available relevant data about the use of U42 Token smart contract methods. Applications can integrate directly with the token by monitoring for events that are of interest to them. For example an application can monitor for StartProvision in order to see that a service has been acquired by a user. In another example, an application can watch for UpdateService to see prices changed on another application's services.

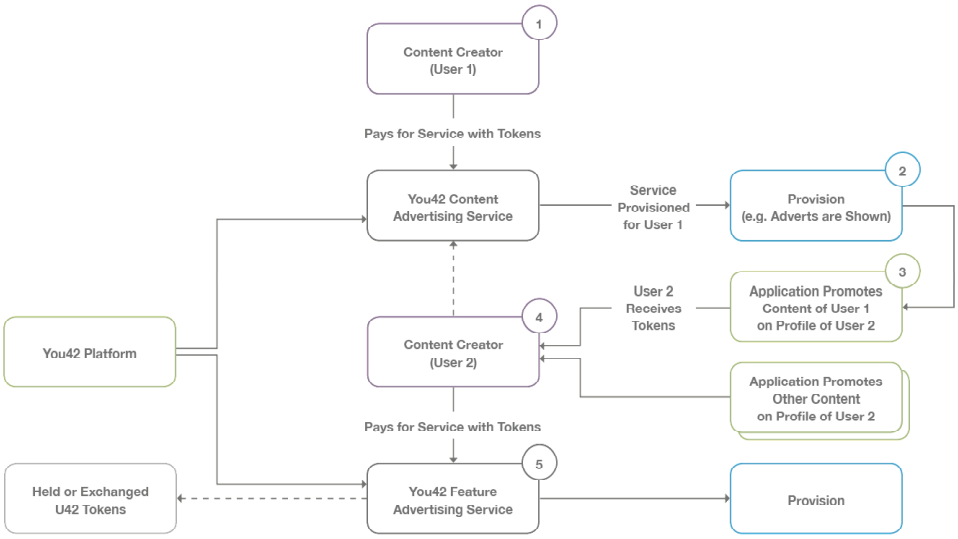
The Token Specification includes a list of events created by the methods of the U42 Token smart contract. Each contract method lists the events that are called on successful use. Note that the transferToService, transferToSimpleService, transferBecauseOf and transferBecauseOfAggregate methods all create ERC-20 standard Transfer events in addition to the U42 Token-specific events defined in the Token Specification.

Protection Against Erroneous Transfers

The U42 Token smart contract provides methods such as transferToService and transferToSimpleService that can be called by users (e.g., content creators, other application users) and expect information about the application and service being provisioned. The behaviour of the smart contract is such that if a transferToService or transferToSimpleService method is called without valid service information (e.g., the service identifier doesn't exist, doesn't match the application or has been removed) then the method call will fail and no tokens will be transferred. This provides an additional level of protection against human error in the use of U42 Token applications.

Note that the application reference field is considered a data>event logging mechanism only, it does not provide a means to verify valid application references.

Examples of Use

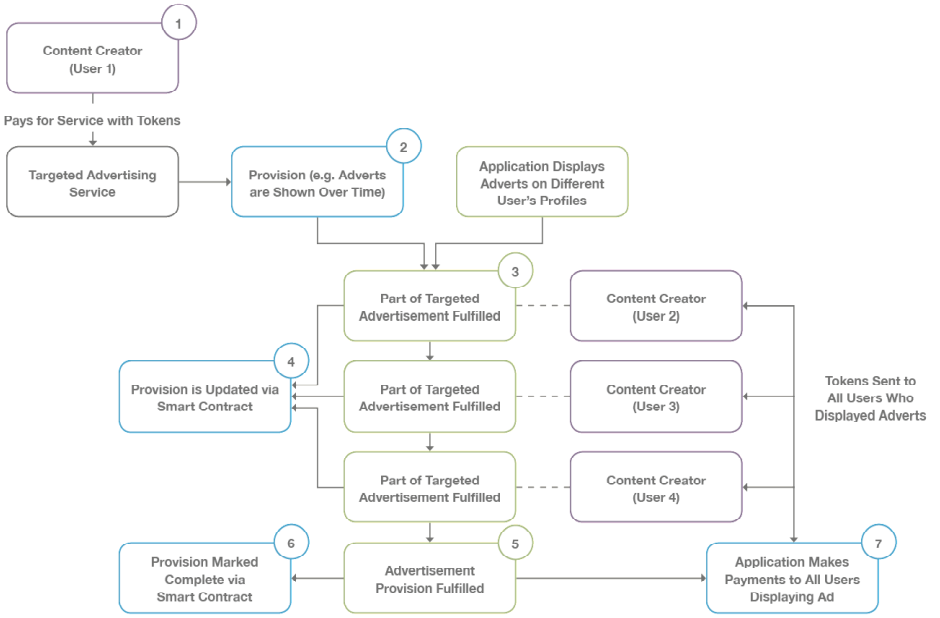


Multiple Service Use – You42 platform
Diagram 2: You42 platform provides multiple advertising services

In this example, the You42 platform provides several advertising services – those that can be displayed on the profiles and content of other users and those that are "feature" advertisements, e.g., displayed on the homepage of the You42 platform.

1. A user purchases an advertising product with tokens
2. That service is provisioned and advertisements are shown on the platform
3. The application keeps track of which users display those advertisements and remunerates User 2 accordingly
4. User 2 receives tokens as a result of advertisements being shown next to their content (including from User 1 above)
5. User 2 uses those tokens to purchase a different service (feature advertising) – they could also purchase the same (content advertising) service or hold the tokens

Examples of Use



Provision Updated Services and Linked Transfer
Diagram 3: Targeted advertising service with tracked usage

Example application provides a targeted advertising service that tracks provision of credited advertisements via the U42 Token smart contract and on completion makes a single set of payments to users who showed that advertisement.

1. A user purchases targeted advertising services

2. The service is provisioned and the application sets up an advertising product that will update the U42 Token smart contract every time part of that provision is used

3. Advertisements are shown on the profiles of content creators and provision updates are sent to the U42 Token smart contract

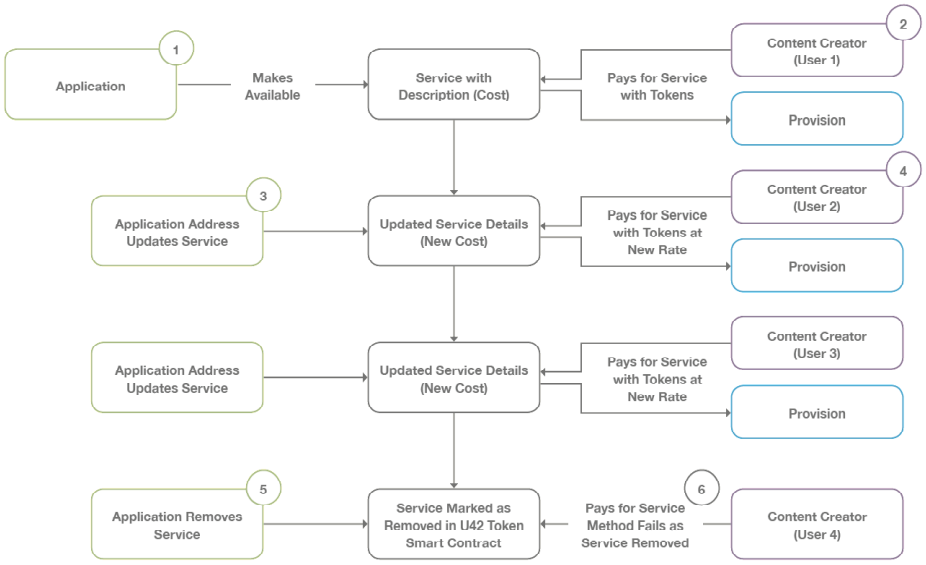
4. The U42 Token smart contract updates the remaining credits for that provision

5. Once the advertisement inventory (credits) is used up the application completes the provision and makes payments

6. The U42 Token smart contract marks the provision as complete and optionally applies any due refund (e.g., where not all advertising credits were used up)

7. The application triggers a token transfer to each of the users that displayed the advertisement using the U42 Token smart contract

Examples of Use



Dynamic Service Availability
Diagram 4: An application that updates the cost of a service and ultimately removes it

In this example an application makes a service available at a specific cost (in tokens/credits) and then updates that cost over time. Ultimately the application removes the service via the U42 Token smart contract remove method and future attempts to provision that service fail (and no tokens are transferred).

1. Application makes a service available via the normal means (note that it can choose to use update addresses – see section “Delegated Security Model”)

2. A user provisions that service at the advertised rate

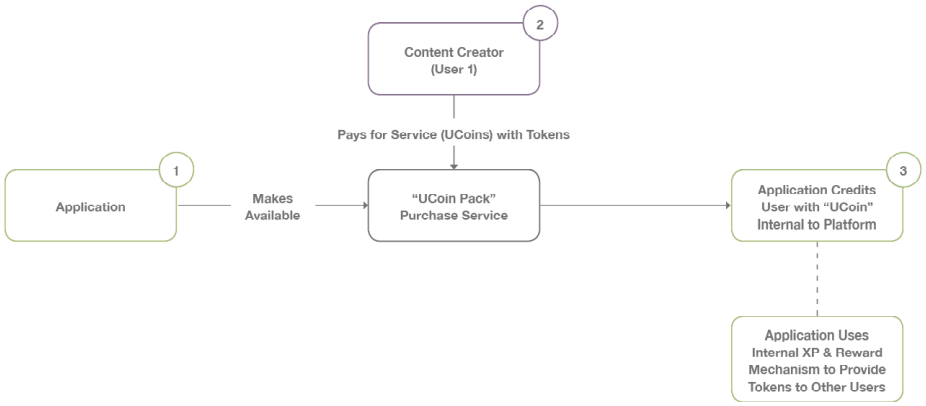
3. The application updates the cost of the services

4. Another user provisions the service at the new rate (note that an attempt to provision the service with the incorrect amount of tokens will fail)

5. Ultimately the application removes the service as it is no longer available

6. When a user attempts to provision the service it will fail as it has been removed (note also that no new services can be created for this application with the same service identifier)

Examples of Use



Simple Services

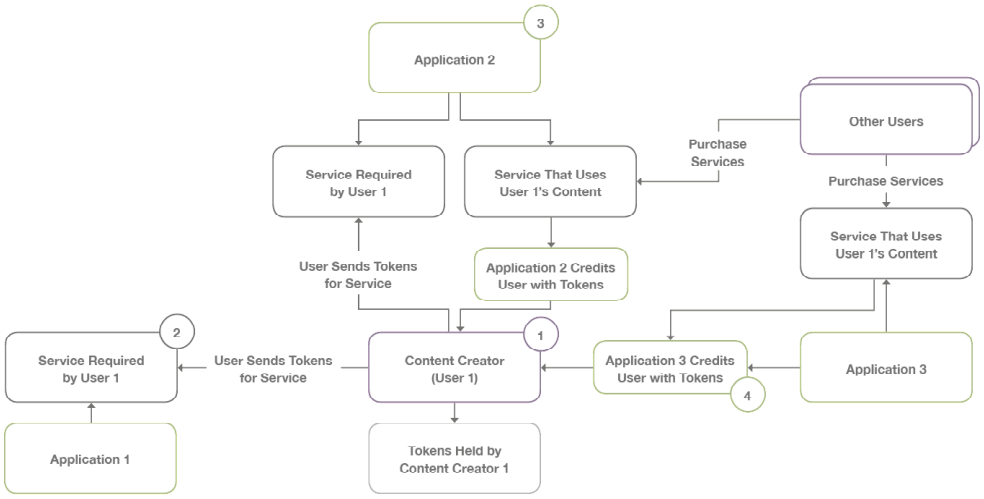
Diagram 5: A Simple "UCoin pack" purchase in the You42 platform

In this example a user purchases a "UCoin pack" from the You42 platform. This is presented by the application as a simple service as it has no credit mechanism and does not require a provision start & end. The application implements something internally to provide a flow of tokens back to other users.

1. Application makes the simple service available via the U42 Token smart contract
2. A user purchases a UCoin pack at its advertised rate (note that simple services can have their information and rates changed as per the previous example)
3. The application provides an internal mechanism to credit the user's UCoin pack purchase

Note that the application could provide an internal mechanism to credit other users with tokens via the standard transfer method. The application would not be able to make a transfer that referenced the service as there is no provision information for a simple service.

Examples of Use



Multiple Applications

Diagram 6: A user interacts with different applications

In this example a user interacts with different applications to provision services and receive payments for its contribution to the services provided. Depending on the requirements of the user these may change over time. The U42 Token is at the center of the user's service interaction.

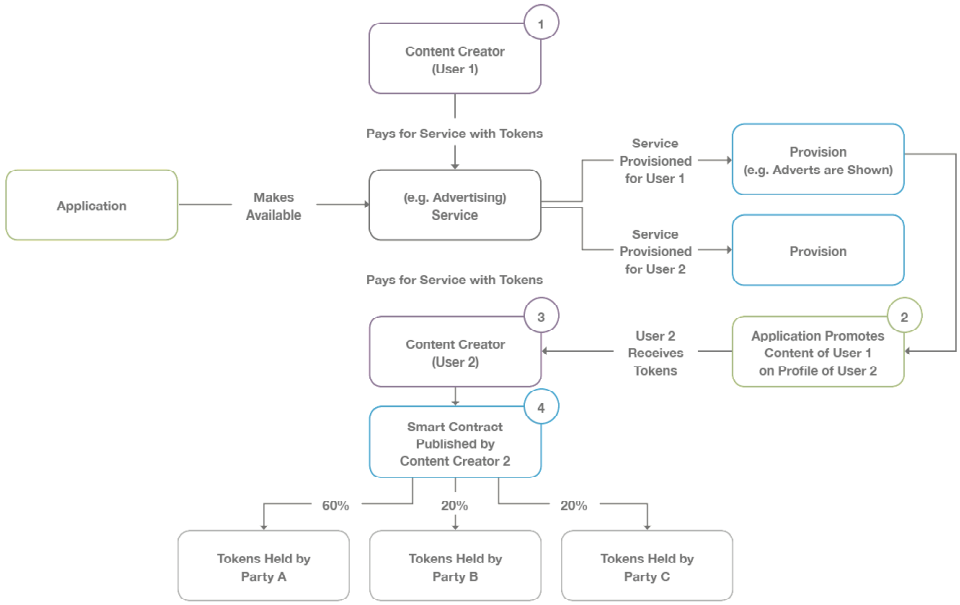
1. The user interacts with different applications and services

2. Some applications provide services that the user requires but don't provide services that the user is paid to participate in

3. Some applications provide services to the user as well as providing services that benefit the user, e.g., advertising alongside content as per previous examples

4. Where applications leverage the user's content for multiple provisions the user can receive tokens as one-off transfers or aggregated transfers

Examples of Use



Integrating Smart Contracts with Application Services (for Content Creators)
Diagram 7: A content creator uses a smart contract to distribute received tokens amongst different parties

In this example a content creator uses a smart contract to distribute received tokens amongst different contributing parties according to an example 60/20/20 royalties split. Note that the user only needs to specify the smart contract address instead of a wallet address and that the application described does not need to be aware of the content creator's specific smart contract details or externally-agreed royalty distribution.

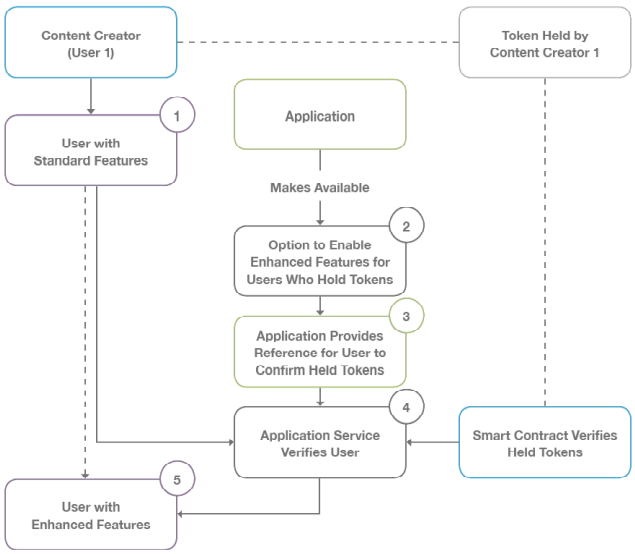
1. A content-creating user of an application pays for an advertising service with U42 Tokens

2. The application tracks which users displayed advertisements for that service and remunerates them with U42 Tokens

3. A different content-creating user (user 2) receives tokens which are sent to the address specified. In this case the address is a smart contract on the Ethereum network

4. The smart contract deployed by the content creator distributes the tokens accordingly, in this example using a 60/20/20 split between three parties

Examples of Use



Making Features Available Exclusively to Token Holders
Diagram 8: A user receives enhanced features by confirming that they hold tokens

In this example an application makes available enhanced features to users that can demonstrate they hold U42 Tokens. This is done without a need for transferring tokens.

1. An existing or new user receives a standard set of features in the application

2. The application makes known an enhanced set of features for users that can confirm they hold U42 Tokens

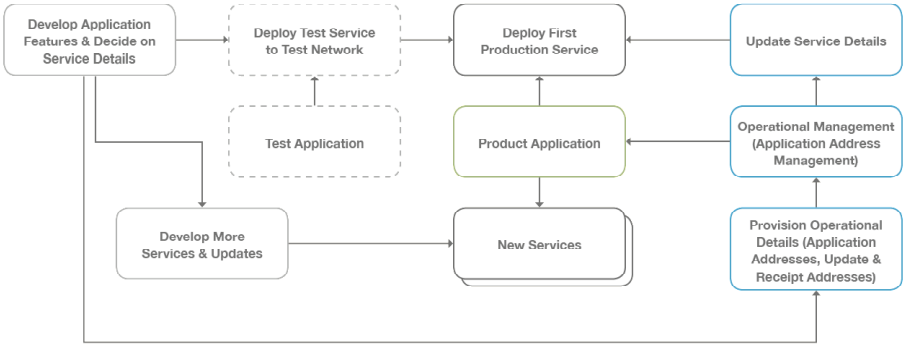
3. A user that wishes to access those enhanced features can request a reference to be confirmed by them using the U42 Token smart contract

4. The application sees the confirmation of the reference via the U42 Token smart contract

5. The application upgrades the user's account with enhanced features

Token Application Development

The U42 Token specification as included in Appendix A of this document is published and updated on the You42 website. The deployed U42 Token smart contract code is available via the linked git repository, along with related technical and deployment details.

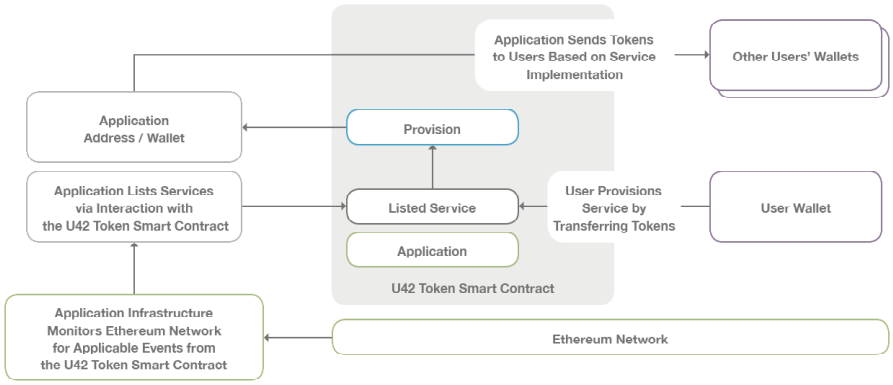


Application Development Summary
Diagram 9: typical application development cycle

Application implementers will typically do at least:

- Application development – the development of applications that provide services compatible with the U42 Token smart contract
- Deployment of test services – see section "Testing Applications with the U42 Token"
- Provision operational details (e.g., production addresses, processes and application-specific smart contracts if required)
- Deployment of production services with operational details
- Update service details – see section "Delegated Security Model"
- Develop additional application services and update existing services

Token Application Development



Structure of a U42 Token Application
Diagram 10: Structure of a U42 Token application

Application implementers:

- Make services available via the U42 Token smart contract
- Monitor the network for events relating to their services
- Receive tokens sent by users to provision services
- Send tokens to users that are involved in the provision of services

Token Application Development

Testing Applications with the U42 Token

For the purpose of testing U42 Token applications, developers can request tokens from the U42 test token faucet on the Ethereum Ropsten test network. The contract address for the token faucet is published on the You42 website.

The test token faucet smart contract acts as a custodian of test U42 tokens and is available for application developers to obtain test tokens to be used solely on the Ethereum Ropsten test network. It issues tokens to specified addresses. Note that a specific amount of tokens cannot be requested and that the faucet smart contract may change the amount of tokens issued based on the use of the faucet.

requestTokens (
    address _sendToAddress
) returns bool success
Sends test U42 tokens to the specified address.

_sendToAddress the address to which tokens should be sent. This can be the same address as the message sender.

Returns true on successful issue of tokens.

If there are no test tokens available the method may return false. Note that the method is intended to be used occasionally by application developers who wish to test token features on the test network and as such may decline repeated requests in a short space of time.

getTokenIssueAmount (
) returns uint256 tokenIssueAmount

Returns an approximate amount of tokens issued to calls to requestTokens based on recent usage of the test token faucet. When called before or after requestTokens the value returned by this method should be similar to the amount of tokens transferred by the requestTokens method.

Appendix A: U42 Token Specification

This specification is intended to fully implement the Ethereum ERC-20 Token Standard (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md) as updated at https://github.com/ethereum/EIPs/commit/7038c5f9b9a4845ee1bf5301c2b0ee800ec181e1.

The core ERC-20 methods name, symbol, decimals, totalSupply, balanceOf, transfer, transferFrom, approve and allowance are implemented as described in the standard and application implementers should be aware of their limitations and risks, especially in regard to supported wallets and interfaces and the risks associated with sending tokens to non-compatible addresses. The You42 Token-specific methods rely on published service specifications, e.g., transferToService, is reliant upon a published application service via listService. This approach significantly reduces the risk of erroneous token transfers when using services from application implementers.

Applications & Addresses

Applications define an address from which they interact with the smart contract. An application could have many application addresses, with each making different services available with different receiving addresses.

Applications will mange a set of (typically one for simple applications) service addresses, from which they make services available and update their details (e.g., token/credit cost). Service identifiers cannot be reused once removed in order to reduce the likelihood of user errors. Each application address can specify ~4m service identifiers.

Applications can additionally specify receipt addresses and update addresses when listing services. Receipt addresses allow for tokens to be sent to an address different from that which created (listed) the service. Update addresses can be specified by the application address to allow for service changes (apart from receipt address) and provision updates.

Application address

Used to list a service. Is the effective owner of the service. The application address is allowed to perform the same functions as any listed update addresses, and can also change the update addresses.

The application address is the default recipient of tokens sent to the service via transfers. The application address can specify a different receipt address to receive tokens. Only the application address can update the receipt address.

Receipt address

A receipt address is intended solely to receive tokens sent to a transfer method (e.g. transferToService, transferToSimpleService).

A receipt address can be a smart contract address that implements a withdraw function to provide automated disbursement of tokens (e.g., via the transfer method).

Update address

Update addresses are intended to provide a lower risk interface mechanism for application services, e.g., where updates to a service description are automated in other systems.

Update addresses can also be used to update the completion of provisioned services. It is anticipated that update addresses would be used alongside receipt addresses to provide a multi-party system of application service creation, updates and disbursements based on the requirements of a specific application.

An update address can also remove a service.

Methods

listService (
uint256 _serviceId,
string _serviceDescription,
uint256 _tokensPerCredit,
uint256 _maxCreditsPerProvision,
address[] _updateAddresses,
address _receiptAddress
) returns bool success
Lists a service available for this application at the sender's address.

_serviceId is a unique numeric identifier for this service. Must be >0 and not already used by this application address.
_serviceDescription is a human-readable string describing the service.
_tokensPerCredit the cost in tokens of each of the application service's credits. Must be >0.
_maxCreditsPerProvision is the total number of credits available for this service. 0 indicates no limit.
_updateAddresses[] an optional (can be empty) list of addresses that are allowed to update this service
_receiptAddress the address tokens should be credited to when a user transfers them to a service

To provide a service with a fixed cost the tokensPerCredit should be passed as the cost (in tokens) and maxCreditsPerProvision should be passed as 1. Note that this can also be achieved with the listSimpleService method, but this assumes that the service is consumed at the point of provisioning.

Fires a NewService event.

Also implemented as listService (serviceId, serviceDescription, tokensPerCredit, maxCreditsPerProvision, updateAddresses) □ passes the sender's address as the receiptAddress

listSimpleService (
uint256 _serviceId,
string _serviceDescription,
uint256 _tokensRequired,
address[] _updateAddresses,
address _receiptAddress
) returns bool success
Creates a service with no credit relationship that doesn't return a provisionId when a user sends tokens with transferToSimpleService.

Methods

_serviceId is a unique numeric identifier for this service. Must be >0 and not already used by this application address (for simple or non-simple services).
_serviceDescription is a human-readable string describing the service.
_tokensRequired the effective cost of the service. Must be >0.
_updateAddresses[] an optional (can be empty) list of addresses that are allowed to update this service
_receiptAddress the address tokens should be credited to when a user transfers them to a service

This is primarily in place to ensure a low (compute) cost alternative for simple services. Applications could still implement listService with maxCreditsPerProvision of 1 and use the returned provisionId (and later rely on provisionStarted, provisionUpdated) by transferToService.

Fires a NewService event.

Also implemented as listSimpleService (serviceId, serviceDescription, tokensRequires, updateAddresses) ⟶ passes the sender's address as the receiptAddress

getServicesForApplication (
    address _applicationAddress
) returns uint32[] serviceIds
Returns a list of service identifiers currently active for this application address.

_applicationAddress the address of the application for which a list of services is required.

View function (does not modify state).

getRemovedServicesForApplication (
    address _applicationAddress
) returns uint32[] serviceIds
Returns a list of a service identifiers that have been removed from this address.

_applicationAddress the address of the application for which a list of removed services is required.

View function (does not modify state).

Methods

getServiceInformation (
address _applicationAddress,
uint32 _serviceId
) returns bool isSimple, uint256 tokensPerCredit, uint256 maxCreditsPerProvision, address[] updateAddresses, address receiptAddress
Returns information about a service.

_applicationAddress the address of the application
_serviceId the identifier of the service. Must be a service listed by the application and not removed.

View function (does not modify state).
For simple services, tokensPerCredit is the effective "tokensRequired" cost of the service.

updateServiceDescription (
address _targetApplicationAddress,
uint32 _serviceId,
string _serviceDescription
) returns bool success
Used to update a service description.

_targetApplicationAddress the address of the application.
_serviceId the identifier of the service to be updated. Must be a valid, non-removed service for this application.
_serviceDescription the new service description.

Can be called by the application address or an update address. When called by the application address it must pass its own address for _targetApplicationAddress.

Fires the ServiceChanged event.

updateServiceTokensPerCredit (
address _targetApplicationAddress,
uint32 _serviceId,
uint256 _tokensPerCredit
) returns bool success
Used to update the tokens per credit cost of a service.

Methods

_targetApplicationAddress the address of the application.
_serviceId the identifier of the service to be updated. Must be a valid, non-removed service for this application.
_tokensPerCredit the new tokensPerCredit value. Must be >0.

Can be called by the application address or an update address. When called by the application address it must pass its own address for _targetApplicationAddress.

Fires the ServiceChanged event.

updateServiceMaxCreditsPerProvision (
address _targetApplicationAddress,
uint32 _serviceId,
uint256 _maxCreditsPerProvision
) returns bool success
Used to update the max credits per provision of a service.

_targetApplicationAddress the address of the application.
_serviceId the identifier of the service to be updated. Must be a valid, non-removed service for this application.
_maxCreditsPerProvision the new maxCreditsPerProvision value. Must be >0.

Can be called by the application address or an update address. When called by the application address it must pass its own address for _targetApplicationAddress.

Fires the ServiceChanged event.

removeService (
address _targetApplicationAddress,
uint32 _serviceId
) returns bool success
Used to remove a service. Removed service identifiers can't be reused by the same application.

Methods

_targetApplicationAddress the address of the application.
_serviceId the identifier of the service to be updated. Must be a valid, non-removed service for this application.

Can be called by the application address or an update address. When called by the application address it must pass its own address for _targetApplicationAddress.

Removed services will not return information from getServiceInformation, but they can be listed by getRemovedServicesForApplication.

Fires the ServiceRemoved event.

changeServiceReceiptAddress (
uint32 _serviceId,
address _receiptAddress
) returns bool success
Change the receipt address of a service.

_serviceId the identifier of the service to be updated. Must be a valid, non-removed service for this application (the message sender).
_receiptAddress the new receipt address.

If an application wishes to remove a previously set receipt address (and revert to the default, i.e., itself) it can pass its own address to the changeServiceReceiptAddress method.

Can only be called by application address.

Fires the ServiceChanged event.

changeServiceUpdateAddresses (
uint32 _serviceId,
address[] _updateAddresses
) returns bool success
Change the allowed update addresses for a service.

_serviceId the identifier of the service to be updated. Must be a valid, non-removed service for this application (the message sender).
_updateAddresses the new list of authorized update addresses.

Note that the entire updateAddresses list must be updated.
Can only be called by application address.
Fires the ServiceChanged event.

Methods

transferToService (
address _applicationAddress,
uint32 _serviceId,
uint256 _tokenValue,
uint256 _credits,
uint256 _applicationReference
) returns uint256 provisionId
Used to acquire a service from an application by transferring tokens.

_applicationAddress the address of the application.
_serviceId the service identifier. Must be a valid non-removed service listed by that application.
_tokenValue the token cost of the service. Must be equal to _credits * {listed tokensPerCredit}. Must be <= balance of tokens for message sender (user).
_credits the number of credits of this service required by this user. Must be >0 and <maxCreditsPerProvision.
_applicationReference optional additional field to be used by an application when requesting service purchases.

Creates a provisionId (uint256) and returns it as a result. Internally tracked to include application, service, tokensPerCredit, credits, applicationReference, user). Note that it is up to an application to determine how to associate a token transfer with a (presumed application) user. This could be done either with the message sender (published in provisionStart and transfer) or with the _applicationReference parameter.

Fires StartProvision and Transfer events.

transferToSimpleService (
address _applicationAddress,
uint32 _serviceId,
uint256 _tokenValue,
uint256 _applicationReference,
uint256 _multiple
) returns bool success
Used to acquire a simple service from an application by transferring tokens.

Methods

_applicationAddress the address of the application.
_serviceId the service identifier. Must be a valid non-removed service listed by that application and created using listSimpleService.
_tokenValue the token cost of the service. Must be equal to _multiple * {listed tokensRequired}. Must be <= balance of tokens for message sender (user).
_applicationReference optional additional field to be used by an application when requesting service purchases.
_multiple the number of this simple service required.

Note that this does not create a provision, it is assumed that the service is consumed at the point of use.

Note that it is up to an application to determine how to associate a token transfer with a (presumed application) user. This could be done either with the message sender (published in provisionStart and transfer) or with the _applicationReference parameter.

Fires a CompleteSimpleProvision and Transfer event.

Also available as transferToSimpleService (_applicationAddress, _serviceId, _tokenValue, _applicationReference) ⟶ passes 1 as _multiple

transferBecauseOf (
address _to,
uint256 _value,
address _applicationAddress,
uint32 _serviceId,
uint256 _provisionId
) returns bool success
Transfers tokens to an address making reference to a service provision.

_to the address to send tokens to.
_value the amount of tokens to send.
_applicationAddress must be either the address of the sender, or an address for which the sender is listed as the receiptAddress (i.e. such that a smart contract deployed at receiptAddress can send funds on behalf of an application/service). Must have balance >=_value.
_serviceId the identifier of the originating service. Can be 0 (indicates no linked provision).
_provisionId the identifier of the associated provision. Can be 0 (indicates no provision linked, or sent as a result of a simple service).

Methods

Note that this is an optional method for application implementers working with listed services and acquired provisions. Tokens can also be sent with the standard transfer method, or not at all.

This method will fail (returning false and not transferring tokens) if the specified _provisionId does not relate to the specified _serviceId and/or the _serviceId does not relate to the specified _applicationAddress.

Fires a TransferBecauseOf event.

transferBecauseOfAggregate (
address _to,
uint256 _value,
address _applicationAddress,
uint32 _serviceId,
unit256[] _provisionIds,
uint256[] _tokenAmounts
) returns bool success

Transfers tokens to an address making reference to a service as a result of several provisions. It is assumed that application implementers only use this method for provisions of the same service.

_to the address to send tokens to.
_value the amount of tokens to send. Must be equal to sum of _tokenAmounts.
_applicationAddress must be either the address of the sender, or an address for which the sender is listed as the receiptAddress (i.e. such that a smart contract deployed at receiptAddress can send funds on behalf of an application/service). Must have balance >=_value.
_serviceId the identifier of the originating service. Can be 0 (indicates no linked provision).
_provisionIds a list of the identifiers of the associate provisions.
_tokenAmounts a list of token amounts for each provision. Length must be equal to length of _provisionIds.

Note that this is an optional method for application implementers working with listed services and acquired provisions. Tokens can also be sent with the standard transfer method, or not at all.

This method will fail (returning false and not transferring tokens) if the specified _provisionIds do not relate to the specified _serviceId and/or the _serviceId does not relate to the specified _applicationAddress.

Fires a TransferBecauseOfAggregate event.

Methods

getProvisionCreditsRemaining (
_uint256 provisionId
) returns uint256 credits
Returns number of remaining credits for provision.

_provisionId the identifier of the provision for which the number of remaining credits is required.

View function (does not modify state).

getProvisionService (
_uint256 provisionId
) returns uint32 serviceId
Returns service identifier for provision.

_provisionId the identifier of the provision for which the serviceId is required.

View function (does not modify state).

getProvisionApplicationAddress (
_uint256 provisionId
) returns address applicationAddress
Returns application address for provision.

_provisionId the identifier of the provision for which the application address is required.

View function (does not modify state).

updateProvision (
address _applicationAddress,
uint32 _serviceId,
uint256 _provisionId
uint256 _creditsRemaining
) returns bool success

Updates the provision with the number of credits remaining.

Methods

_applicationAddress the address of the application for this provision's service. Sender must be the _applicationAddress or a listed updateaddress for the specified application/service.
_serviceId the serviceId of the provision. Must be the serviceId obtained by getProvisionService.
_provisionId the identifier of the provision to update.
_creditsRemaining the number of credits remaining for this provision. Must be >0 (completeProvision should be used when remaining credits is 0).

Can be called by application or update addresses of that application.

Fires UpdateProvision event.

completeProvision (
address _applicationAddress,
uint32 _serviceId,
uint256 _provisionId
uint256 _creditsRemaining
) returns bool success

Completes the provision. Completed provisions will not return information from getProvision* methods. Any credits remaining will be sent back to the original acquiring user.

_applicationAddress the address of the application for this provision's service. Sender must be the _applicationAddress or a listed updateAddress for the specified application/service.
_serviceId the serviceId of the provision. Must be the serviceId obtained by getProvisionService.
_provisionId the identifier of the provision to update.
_creditsRemaining the number of credits remaining for this provision. Must be >=0. If >0, creditsRemaining * tokensPerCredit (at time of provision) will be credited back to original user (as per transferToService).

Can be called by application or update addresses of that application address.

Note that completeProvision cannot be called for simple services.

Fires CompleteProvision event.

Methods

confirmReference (
address _applicationAddress
uint256 _applicationReference,
uint256 _senderTokensGreaterThan
) returns bool success
Used to confirm an application reference with a minimum number of held tokens. Does not transfer tokens.

_applicationReference a reference supplied to the caller by an application.
_applicationAddress the address of the application that this reference relates to.
_senderTokensGreaterThan the number of held tokens in order to confirm the reference.

Verifies that the sender of the message has more than _senderTokensGreaterThan token balance. Used to verify application references without transferring tokens.

Note that specifying 0 as the _senderTokensGreaterThan simply verifies that the sender holds any number of tokens.

Is considered unsuccessful and returns false if the sender does not have greater than _senderTokensGreaterThan tokens.

Fires the ReferenceConfirmed event when successful.

name (
) returns string {Token Name}
View function. Returns the name of the token, e.g., “You42 Token” for main net. As per ERC-20 standard.

symbol (
) returns string {Token Symbol}
View function. Returns the symbol of the token, e.g., “Y42” for main net.

decimals (
) returns uint8 {Decimal precision}
View function. Returns the decimal precision of the token, e.g., 8. As per ERC-20 standard.

totalSupply (
) returns uint256 {Total supply}
View function. Returns total supply of token. As per ERC-20 standard.

Methods

balanceOf (
address _owner
) returns uint256 balance
View function. Returns balance of account for owner. As per ERC-20 standard.

_owner address of account to query balance of.

transfer (
address _to,
uint256 _value
) returns bool success
Transfers tokens to address from message sender. As per ERC-20 standard.

_to address to transfer tokens to.
_value amount of tokens to be transferred. Must be <= balance of tokens for sender. Accepts 0 as per standard.

Fires Transfer event.

transferFrom (
address _from,
address _to,
uint256 _value
) returns bool success
Transfers tokens from one address to another. As per ERC-20 standard. **To be implemented for compatibility with base standard, not intended for use in normal You42 application/service scenarios.**

_from address to transfer tokens from. Message sender must be approved via approve.
_to address to transfer tokens to.
_value amount of tokens to transfer. Must be <= balance of tokens from _from address.

Fires Transfer event.

approve (
address _spender,
uint256 _value
) returns bool success
Authorizes addresses to transfer tokens for other addresses. As per ERC-20 standard. **To be implemented for compatibility with base standard, not intended for use in normal You42 application/service scenarios.**

Methods

_spender address to authorize for transfer from this address.
_value total amount to allow for transfer.

Fires Approval event.

allowance (
address _owner,
address _spender
) returns uint256 remaining

View function. Returns amount one address is authorized to spend for another. As per ERC-20 standard. **To be implemented for compatibility with base standard, not intended for use in normal You42 application/service scenarios.**

_owner address _spender is authorized to spend from.
_spender address authorized.

Events

Transfer (
address indexed _from,
address indexed _to,
uint256 _value
)
As per ERC-20 standard. Also fired from service transfers.

Approval (
address indexed _owner,
address indexed _spender,
uint256 _value
)
As per ERC-20 standard. Also fired from service transfers.

NewService (
address indexed _applicationAddress,
uint32 _serviceId
)
Indicates a new service for application address. Can be used to get service information from get* methods.

ServiceChanged (
address indexed _applicationAddress,
uint32 indexes _serviceId
)
Indicates a change to a service. Triggered for change* and update* events. Can be used to get service information from get* methods.

ServiceRemoved (
address indexed _applicationAddress,
uint32 indexes _serviceId
)
Indicates removal of a service.

Events

StartProvision (
address indexed _applicationAddress,
uint32 indexed _serviceId,
address indexed _userAddress,
uint256 _serviceCredits,
uint256 _tokensPerCredit,
uint256 _provisionId,
uint256 _applicationReference
)
Indicates start of a service provision. Not fired for simple services.

UpdateProvision (
address indexed _applicationAddress,
uint32 indexed _serviceId,
uint256 indexed _provisionId,
uint256 _creditsUsed
)
Indicates update of service credit usage. Not all applications will update provisions. Not fired for simple services.

CompleteProvision (
address indexed _applicationAddress,
uint32 indexed _serviceId,
uint256 indexed _provisionId,
uint256 _creditsUsed
)
Indicates completion of provision and final update of service credit usage. Not fired for simple services.

CompleteSimpleProvision (
address indexed _applciationAddress,
uint32 indexed _serviceId,
address indexed _userAddress,
uint32 _multiple,
uint256 _applicationReference
)
Indicates start and completion of a simple service. Note that there's no provisionId for simple services.

Events

TransferBecauseOf (
address indexed _to,
address indexed _forApplicationAddress,
uint256 indexed _provisionId,
address _from,
uint32 _serviceId,
uint256 _value
)
Transfer associated with a specific provision. Indexed field count is at 3 as per Solidity limitation. Note that there is no expectation that an application uses transferBecauseOf vs transfer vs not crediting transfer balances at all.

TransferBecauseOfAggregate (
address indexed _to,
address indexed _forApplicationAddress,
uint32 indexed _serviceId,
address _from,
uint256 _value,
uint256[] _provisionIds,
uint256[] _tokenAmounts
)
Transfer associated with an aggregate provision. Indexed field count is at 3 as per Solidity limitation. Note that there is no expectation that an application uses transferBecauseOfAggregate or transferBecauseOf vs transfer vs not crediting transfer balances at all.

ReferenceConfirmed (
address indexed _applicationAddress
uint256 indexed _applicationReference,
address indexes _confirmedBy,
uint256 _senderTokensGreaterThan
)
Indicates the _applicationReference for _applicationAddress application was confirmed by address _confirmedBy that held more than _senderTokensGreaterThan tokens at the time of the confirmation.

Token Deployment Parameters

Development and test

For the purposes of testing and token development during the launch of the You42 platform, the token will be deployed to the Ropsten test network with the following parameters. Note that these have been chosen to be deliberately different from the public test and main network token as they are only for development purposes. A public test faucet will not be provided for this test token.

- **Token Name:** "Dev 54 Token"
- **Token symbol:** "D54"
- **Token supply:** (same as for main network)
- **Decimal precision:** (same as for main network)

Public test

For testing integrated services with the U42 Token, a version of the implemented U42 Token smart contract will be deployed to the Ropsten test network with the following parameters. See Testing Applications with the U42 Token for a description of the test faucet available to application developers to request test tokens to be used for testing applications with the U42 Token smart contact.

- **Token Name:** "U42 Test Token"
- **Token symbol:** "T42"
- **Token supply:** (same as for main network)
- **Decimal precision:** (same as for main network)

Main network

The U42 Token as published on the main Ethereum network.

- **Token Name:** "U42 Token"
- **Token symbol:** "U42"
- **Token supply:** 525,000,000
- **Decimal precision:** 8

Definitions

You42 Platform: You42 is the world's premiere social entertainment platform designed around an ecosystem where creators can engage with fans like never before. You42 is a web application built in HTML5 and accessible through any browser, on any device, anywhere.

U42 Token: Issued and transacted on a public blockchain, the U42 Token is a service token that is used to purchase advertising in a per-use or per-display model based on a rate set by the platform at the time of purchase. The token is based on the Ethereum ERC-20 token standard.

Creator: An individual, group of individuals or brand who creates or owns content to share on their own You42 profile or page to generate views, listens, interaction and/or revenue from their audience.

User: Any individual who has as a profile on the You42 platform.

Advertiser: A creator or brand who promotes a product, content, or event on the You42 platform.

UCoin: UCoins are You42's internal value exchange. UCoin can be earned for specific activities in the platform or can be purchased via the U42 token. As a user, the UCoin can be used to access premium content or tip creators and as a creator are utilized to buy promotional advertising.

XP: experience points are earned throughout the site for users participating: sharing, posting, etc. As users 'level up' in XP, they can be rewarded via the U42 Token upon hitting benchmarks.

Socialfeed: The newsfeed that runs through the center of the You42 platform allowing users and creators to socialize, share content and hang out

Elements: Small or unique pieces of content that can be valuable to fans of a content creator such as music stems, directors commentary or artwork concepts

Moments: Pieces of content that (mostly) users upload onto the site - includes photos & pictures, music and videos

Connect

You42 Inc.
202 Tribble Gap Rd.
Suite 300
Cumming, GA 30040

Email

info@u42.io

Website

u42.io