# Identity Architecture (revision 5)

Revised Draft June 18, 2012

Dick Hardt

[dickhardt@gmail.com](mailto:dickhardt@gmail.com)

Contributors:

Hugh Cumming

[Hugh.cumming@securekey.com](mailto:Hugh.cumming@securekey.com)

Troy Ronda

[Troy.ronda@securekey.com](mailto:Troy.ronda@securekey.com)

Mike Varley

[Mike.varley@securekey.com](mailto:Mike.varley@securekey.com)

Peter Watkins

[Peter.watkins@gov.bc.ca](mailto:Peter.watkins@gov.bc.ca)

Patricia Wiebe

[Patricia.wiebe@gov.bc.ca](mailto:Patricia.wiebe@gov.bc.ca)

# SecureKey open Specification Promise

SecureKey Technologies Inc. ("SecureKey") irrevocably promises not to assert any Necessary Patent Claims against you for making, using, selling, offering for sale, importing or distributing your implementation of the specification listed below (the "Specification"), but only to the extent your implementation conforms to the Specification. This promise is subject to the following. This is a personal promise directly from SecureKey to you, and you acknowledge as a condition of benefiting from it that no SecureKey rights are received from suppliers, distributors, or otherwise in connection with this promise. If you file, maintain or voluntarily participate in a patent infringement lawsuit against a SecureKey implementation of the Specification, then this personal promise does not apply with respect to any implementation of the Specification made or used by you. "Necessary Patent Claims" are those claims of patents owned or controlled by SecureKey that are necessary to implement the required portions of the Specification, and that are described in detail and not merely referenced in the Specification. No other rights except those expressly stated in this promise shall be deemed granted, waived or received by implication, exhaustion, estoppel, or otherwise. Please note that this promise applies only to the version of the Specification listed below. Unless otherwise expressly indicated, this promise does not apply to any other version of the Specification listed below.

Covered Specifications:

“Identity Architecture (revision 5)”

# Abstract

This document describes a high level architecture for delivering public and private sector identity services. An App interacting with a user would like to obtain data about the user from one or more Data Providers. The App directs the user to the Identity Agent. The Identity Agent authenticates the user and prompts the user to authorize release of data from Data Providers. The Identity Agent then asks the Authentication Service to generate a Ticket for the Ticket Service that the Ticket Service transforms into a Ticket specific for each Data Provider. The Data Provider Tickets are then exchanged for user specific data from the Data Providers.

# Table of Contents

GoBC Identity Architecture (revision 5) 1

Abstract 1

Table of Contents 3

1. Roles 4

2. Use Cases 5

3. App Data Flow 9

4. App Registration 11

5. Tickets 12

6. API 14

7. Security Considerations 18

8. Privacy Considerations 18

9. Error Codes 19

10. References 20

# Roles

## User

The person interacting with the App who is the subject of the transaction.

## App

The software running on the local device (smartphone, PC) and or server that the User is interacting with. See the Security Considerations section for local vs server software residency recommendations. Each App has a globally unique identifier in the form of a domain name. Apps trust the Ticket Service.

## Privileged App

An App with special policies from one of more Data Providers such as not requiring User transaction authorization or Passcodes.

## Identity Agent

An application or browser plugin residing locally on the device the User is interacting with that prompts the User to present credentials and authorize transactions. The Identity Agent is invoked by the App when the App wants data about the User from Data Providers.

The Identity Agent acquires credentials from the Authentication Service for making authenticated calls to the Ticket Service.

The Identity Agent and Authentication are tightly bound. The architecture shows them as separate components to illuminate what functionality is local on a device versus on a server.

## Mobile Identity Agent

An Identity Agent running on a smartphone.

## Mobile Identity Agent Locator Service

A service that can communicate directly to a User’s Mobile Identity Agent to request authentication and authorization.

## Authentication Service

The Authentication Service transforms credentials from the User into a Ticket Service specific Ticket containing an identifier for the User known only to the Ticket Service and the Authentication Service. The Authentication Service may provide Tickets to more than one Ticket Service. The Authentication Service has a globally unique identifier. The Identity Agent and Authentication are tightly bound. The architecture shows them as separate components to illuminate what functionality is local on a device versus on a server.

## Ticket Service

The Ticket Service converts Tickets from the Authentication Service into Tickets for Data Providers according to the Ticket Service Policy. The Ticket Service may accept Tickets from more than one Authentication Service and will have a different identifier for a given User for each Authentication Service. The Ticket Service may contain multiple identifiers from the same Authentication Service for the same User. The Ticket Service maintains a table of correlated identifiers for each User. The identifiers maintained by the Ticket Service are known only to the Authentication Service or the Data Provider. The Ticket Service does have identifiers the Data Provider uses when interacting with a User or the credentials the Authentication Service accepts. The Ticket Service maintains the user’s PassCode to verify the User. The Ticket Service trusts the Authentication Service

## Data Provider

The Data Provider is an App that provides data about Users to other Apps. The Data Provider accepts Authorization Tickets and returns Data about the user according to the Data Provider’s Policy. The Data Provider provides a Data Description of what data about a User it may release to a given App.

## Developer

A person with administrative rights to register and manage the App at the Ticket Service and Data Providers.

## Operator

A person operating a counter or mobile identity verification App.

# Use Cases

## Counter Application

This is an App at the counter of a government office, a health care facility or a bank that needs to identify the User to conduct a transaction. The Operator is sitting behind the counter.

1. User approaches counter.
2. Purpose of App is communicated to the User either orally or in writing.
3. User taps card on reader.
4. Operator confirms photo from Data Provider matches User.
5. Operator proceeds with processing.

## Kiosk

This is an App running on a dedicated device providing the User with an automated service.

1. User picks language and initiates transaction.
2. User taps card.
3. User enters PassCode.
4. User authorizes transaction.
5. Kiosk gathers identity data and completes transaction.

## Federated Physical Access Management

This is an App that is controlling physical access to a building, room or storage area. There may be multiple organizations that can authorize Users to have access. Each organization is represented by a Data Provider.

1. User taps card on reader.
2. App gets Tickets for all federated Data Providers.
3. App confirms a Data Provider has granted access to the User.
4. The App allows entry to the User.

## Mobile Identity Verification

This is an App running on the Operator’s smartphone. The smartphone has an NFC reader. The Operator requires identity data about the User to perform a transaction.

1. Operator communicates purpose of App to the User and requests User’s card.
2. User provides Operator their card.
3. Operator taps card on smartphone.
4. Operator confirms photo from Data Provider matches User.
5. Operator proceeds with processing.

## Mobile Application

This is an App the User has installed on their smartphone. The User has also installed and enrolled an Identity Agent on their smartphone. The App is providing functionality where identity data about the User is either desirable or required.

1. User initiates identity transaction with App.
2. App invokes Identity Agent.
3. Identity Agent prompts User to enter PassCode if requested by App.
4. Identity Agent presents User with descriptions from each Data Provider about User data that may be released and prompts User to authorize transaction.
5. User authorizes transaction.
6. Identity Agent acquires Data Provider Tickets and invokes App.
7. App proceeds with functionality.

## One Time PC Web Login with Mobile

The User wants to log into a web App running on a PC with no card reader using the Identity Agent running on the User’s smartphone. The User has installed and enrolled an Identity Agent on their smartphone. The App is providing functionality where identity data about the User is either desirable or required. The User has not used the App recently and does not expect to use the App again soon.

1. User navigates to the web App in the browser on the PC and initiates an identity transaction with App.
2. The App displays a QR code representing a Login URL (the Login URL may also be displayed) and instructs User to scan the QR code.
3. The User opens the Identity Agent on their smartphone and scans the QR code (or types the Login URL into an input field).
4. Identity Agent prompts User to enter PassCode if requested by App.
5. Identity Agent presents User with descriptions from each Data Provider about User data that may be released and prompts User to authorize transaction.
6. User authorizes transaction.
7. Identity Agent acquires Data Provider Tickets and redirects to the Return URL in the Login URL.
8. The App displays a message in the browser on the smartphone that it is completing the login process.
9. The App displays on the PC that the login was successful. The browser window on the smartphone is closed. The App proceeds with functionality.

NOTE: The Login URL is the /ticket API for the Identity Agent and is parsed as if the App had launched the Identity Agent with the URL. The return URL in the Login URL contains a session identifier or token so that the App can bind the session in the PC browser with the smartphone browser.

## Frequent PC Web Login with Mobile

The User wants to log into a web App running on a PC with no card reader using the Identity Agent running on the User’s smartphone. The User has installed and enrolled an Identity Agent on their smartphone. The App is providing functionality where identity data about the User is either desirable or required. The User expects to use the App again in the near future and would like the browser on the PC to remember her so that subsequent logins are quicker.

The initial login is the same experience as 2.6 with the addition that the User has selected an option during login on the PC to remember her.

Subsequent logins proceed as follows:

1. User navigates to the web App in the browser on the PC and initiates an identity transaction with App.
2. The App displays an identifier to the User recognizable to the User as belonging to her. If more than one User has been remembered, those Users are also presented to the User.
3. The User selects her identifier to initiate login. If her identifier is not presented, then she selects an option which takes her to 2.7(B)
4. The App informs the User that her smartphone is being contacted.
5. The User’s Mobile Identity Agent alerts her that an App would like to authenticate.
6. Identity Agent prompts User to enter PassCode if requested by App.
7. Identity Agent presents User with descriptions from each Data Provider about User data that may be released and prompts User to authorize transaction.
8. User authorizes transaction.
9. Identity Agent acquires Data Provider Tickets and redirects to the Return URL in the Login URL.
10. The App displays a message in the browser on the smartphone that it is completing the login process.
11. The App displays on the PC that the login was successful. The browser window on the smartphone is closed. The App proceeds with functionality.

**NOTE**: 2.6(D-I) are the same as 2.7(F-K)

# App Data Flow

# 

The flow illustrated in Figure 1 indicates how an App acquires data about a User from a Data Provider. The specific steps are:

1. The User initiates authentication with the app.
2. The App initiates authentication of the user by calling the Identity Agent with the Ticket Service id (TS\_ID), theAppID, the Data Provider IDs (DP\_IDs) for any Data Providers, the Return URL for the Identity Agent to send the response and any options. See Identity Agent API for details. The Identity Agent may receive the call by scanning a QR code generated by the App.
3. If the App provided any DP\_IDs and authorization is requested, the Identity Agent calls each of the Data Provider’s data description endpoints with the AppID. This may be done in parallel with steps (D) – (H)
4. The Data Provider returns the Data Provider data description (DP\_Description) appropriate for the App. The Identity Agent may cache the App specific DP\_Description per DP\_Description meta data.
5. If the Identity Agent can read NFC cards, the Identity Agent prompts the User to tap their card.
6. The User taps their card
7. If the App requested a PassCode, the Identity Agent prompts the user to enter their PassCode.
8. The user enters their PassCode.
9. If the App requested authorization, the Identity Agent presents the user with the DP\_Descriptions from any Data Providers and prompts the user for authorization.
10. The User agrees to the authorization.
11. The Identity Agent calls the Authentication Service with the TS\_ID and card or other Identity Agent key material. The Identity Agent will NOT send the AppID or DP\_IDs to the Authentication Service.
12. The Authentication Service generates a TS\_Ticket that includes the TS\_ID, the TS\_UserID and a datetime stamp and is encrypted with the Ticket Services key. See Ticket Section for more detail on Ticket encodings.
13. The Identity Agent calls the Ticket Service with the AppID, the ReturnURL, the TS\_Ticket, any DP\_IDs, the PassCode if collected, and an Authorization flag indicating if the user had explicitly granted authorization.
14. The Ticket Service:
    1. verifies the AppID is valid.
    2. verifies the ReturnURL has been registered to the App.
    3. evaluates the Ticket Service policy for the App and any provided DP\_IDs.
    4. verifies the TS\_Ticket is valid, from an authorized Authentication Service and contains a known TS\_UserID.
    5. looks up the matching DP\_UserIDs for the TS\_UserID
    6. if appropriate, generates an Authorization Ticket (AZ\_Ticket) for the AppID and any requested DP\_IDs. Each AZ\_Ticket includes the AppID, the DP\_UserID, authentication & authorization meta data, and a datetime stamp. The AZ\_Ticket is encrypted with the Data Providers key.
    7. The AZ\_Tickets are returned to the Identity Agent.
15. The Identity Agent returns the AZ\_Tickets or an error to the App by redirecting to the ReturnURL. See Errors for more information about errors. The App now knows the App specific DP\_UserID for the user by looking at the App’s AZ\_Ticket.
16. If other AZ\_Tickets are received, the App may call the Data Provider passing the AppID, the DP\_Secret the App has for the Data Provider, and the AZ\_Ticket.
17. The Data Provider:
    1. verifies the AppID and DP\_Secret match.
    2. verifies the AZ\_Ticket and the AppID in the AZ\_Ticket matches the AppID passed in.
    3. verifies the authorization and authentication metadata meets the Data Provider’s policy for the App
    4. evaluates the Data Provider’s policy for the App, the AZ\_Ticket authentication and authorization metadata, and if appropriate returns user data for the DP\_UserID in the AZ\_Ticket.

NOTE: There may be additional implementation dependent calls between the Identity Agent and the Authentication Service for setup and the timing of steps (K) and (L) may be earlier.

# App Registration

Each App has a unique identifier in the form of a domain name. In order to participate in the system, the Apps must register at the Ticket Service and then at any Data Provider’s they wish to acquire data from about the User. To maintain system integrity, the Ticket Service creates App Registration Tickets that are consumed by a given Data Provider that assert the Developer is authoritative for an App.

## App Registration at Ticket Service

1. The Developer navigates to the Ticket Service’s Registration App and clicks a link to login on to the Registration App.
2. The Registration App initiates a standard authentication request with the Developer with his Identity Agent with the Registration App ID.
3. The Developer authenticates and authorizes the transaction with her Identity Agent.
4. The Identity Agent sends the Authentication Ticket and the Registration App ID to the Ticket Service, which then generates a Ticket for the Registration App ID.
5. The Identity Agent redirects back to the Registration App with the Ticket. The Registration App Developer is now authenticated to the Registration App.
6. If the Developer is an administrator for any existing Apps, she can select to administer one of them or create a new App. Otherwise, the Registration App presents the Developer with an App registration form
7. The Developer enters an App ID (hostname?), return URL(s), name, description, and authorization description for the app and agrees to the Ticket Service’s Terms of Service. The Ticket Service checks that there is no other App with that ID and registers the App.

**NOTE**: There is nothing special in (B-E) from a User logging into a regular App.

**TBD**: need mechanism for Developer to add other Developers to manage the App as the Ticket Service is authoritative for who is able to administer an App.

**TBD**: the Ticket Service can provide a subdomain that the Developer can use as an App ID to ease registration for non-web based Apps and hosting of authorization descriptions for simple Apps.

## App Registration at Data Provider

1. The Developer navigates to the Data Provider App Registrator (DPAR) at the Data Provider and enters the App ID they would like to register and clicks a link to initiate registration.
2. The DPAR initiates a standard authentication request with the Developer with his Identity Agent with the App ID being the Data Provider’s ID, and the DP\_IDs being: the Ticket Service Registration App ID, and the App ID entered by the Developer.
3. The Developer authenticates and authorizes the transaction.
4. The Identity Agent sends the Authentication Ticket and the App ID and DP\_IDs to the Ticket Service.
5. The Ticket Service detects that one of the DP\_IDs is the Ticket Service Registration App ID and then knows the other DP\_ID is the App ID being registered (Registration ID). The Ticket Service checks if the App with Registration ID is valid and if the Developer is an administrator for the App. If so, the Ticket Service generates an App Registration Ticket signed with the secret key shared with the Data Provider and containing the Data Provider ID for the Developer and the Registration ID and sends them back to the Identity Agent. **\*see below**
6. The Identity Agent redirects back to the DPAR sending the App Registration Ticket.
7. The DPAR decrypts the App Registration Ticket and knows the Developer is authoritative for the App represented by the Registration ID.
8. The DPAR prompts the Developer to accept the Data Provider’s Terms of Service.
9. The Developer accepts the Data Provider’s Terms of Service.
10. There may be additional interactions between the Developer and the Data Provider depending on the policies of the Data Provider.
11. The DPAR then registers the App at the Data Provider and generates an App Secret for the App to use when making API calls to the Data Provider.

**NOTE**: The only unique aspect of this flow over a standard flow is the Ticket Service is detecting its Registration App ID and generating an App Registration Ticket.

**TBD**: Can this protocol leak information about a User who is a Developer if abused by an App?

**\*** *Another possible flow would be to generate a standard Ticket at the Ticket Service in (E) and for the Data Provider to call a Ticket Service API to get confirmation the Developer is authoritative for the App.*

# Tickets

Tickets are encoded using JWT[1] and encrypted using JWE[2]. This enables Tickets to be passed as URL parameters without encoding issues. Tickets are encrypted with a symmetric key shared between the Ticket Issuer and the Ticket Audience. The Ticket Audience will always be a single party.

## Header

The JWT header will contain the crypto algorithms and the key ID. (recommend the Key ID contain the ids for the two parties followed by a number that is incremented with each new key generation.

## Payload

The JWT payload must include:

1. iat - date/time stamp when Ticket was generated as IntDate
2. iss - ID of party that generated the Ticket (Ticket Issuer)
3. aud - ID of party that Ticket is for (Ticket Audience, Relying Party, RP)
4. prn - directed identifier of User for RP
5. typ - type of Ticket. Valid types are authentication, authorization and registration
6. app.gov.bc.ca - requesting App ID, or the App ID being Registered in the case of an App Registration Ticket
7. auth.gov.bc.ca – object containing Authentication and Authorization meta data about how the User was authenticated and if the User has explicitly authorized the transaction. Object MUST contain the passcode and authorization properties (other properties TBD)

Example Authentication Ticket Payload initiated by app.example.com:

{ “iat”:1300819380

, “iss”:”securekey.com”

, “aud”:”ticket.com”

, “prn”:”8udh36fja”

, “typ”:”authentication”

, “app.gov.bc.ca”:”app.example.com”

, “auth.gov.bc.ca”:

{ “passcode”:true

, “authorization”:true

, “nfc”:true

}

}

Example Authorization Ticket Payload for App app.example.com wanting to access Data Provider data.com:

{ “iat”:1300819450

, “iss”:”ticket.com”

, “aud”:”data.com”

, “prn”:”uY2v5sWm1”

, “typ”:”authorization”

, “app.gov.bc.ca”:”app.example.com”

, “auth.gov.bc.ca”:

{ “passcode”:true

, “authorization”:true

, “nfc”:true

}

# API

## Identity Agent API

This is the API that the Identity Agent presents to Apps. Examples use newlines for legibility and should be ignored.

## /info

Returns information of the Identity Agent such as version, features, and supported Ticket Services. Result parameters:

protocol - a string containing the version of the protocol supported. MUST be 1.0

vendor - a hostname indicating the vendor of the Identity Agent

version - a vendor specific string indicating the version of the Identity Agent

nfc – true|false – indicates if the Identity Agent can read an NFC card

ticket – a list of comma separated Ticket Service identifiers supported by the Identity Agent

### Request

ia://info?return\_url=myapp%3A%2F%2Fia%2Finfo

### Response

app://ia/info

?protocol=1.0

&vendor=securekey.com

&version=beta

&nfc=false

&ticket=ticket.com,newticket.com

## /ticket

Requests one or more Tickets. Request parameters:

app - REQUIRED ,the App ID

ticket – REQUIRED , a Ticket Service ID

language – OPTIONAL, the language the App would like used. Default is en-ca.

passcode – OPTIONAL, set to false if the User is not to be prompted to enter their Passcode. Default is true.

authorization – OPTIONAL, set to false if the User is NOT to be prompted to authorize the transaction. Default is true.

agent\_locate – OPTIONAL, set to true if the User is to be prompted to authorize release of their Mobile Identity Agent and an agent\_location parameter returned. Default is false. Used for streamlining subsequent PC Web logins.

### Request

A sample request for the App with id [myapp.com](https://myapp.com) , Ticket Service [ticket.com](https://ticket.com) and asking for Tickets for Data Providers [data.com](https://data.com) and [moredata.com](https://moredata.com) and a Return URL of app://ia/ticket.

ia://ticket

?app=myapp.com

&ticket=ticket.com

&language=en-ca  
&passcode=false

&authorization=false  
&providers=data.com,moredata.com  
&return\_url=app%3A%2F%2Fia%2Fticket

A sample request that includes asking for the Location URL for the Mobile Identity Agent Locator Service:

ia://ticket

?app=myapp.com

&ticket=ticket.com

&agent\_locate=true  
&providers=data.com,moredata.com  
&return\_url=app%3A%2F%2Fia%2Fticket

### Response

A successful response contains the Data Providers with their respective AZ\_Tickets:

ia://ticket

?data.com=D261…7FCB

&moredata.com=B517…2FA4

A successful response if agent\_location was specified and the request was authorized contains the Location URL in the agent\_location parameter:

ia://ticket

?data.com=D261…7FCB

&moredata.com=B517…2FA4

&agent\_location=http%3A%2F%2Fagent.com%2F…

The Location URL MUST not include query parameters and must contain transaction information to enable the Mobile Identity Agent Location Service to confirm the URL is valid and to locate the Mobile Identity Agent.

The response in the case of an error, for example the user cancelled the transaction:

ia://ticket?error=USER\_CANCELLED

## /event/register

Registers a Return URL for the App and puts the Identity Agent into a mode where cards are read and processed as presented

required for the Federated Physical Access use case.

Request and Response TBD

## /event/delete

Deletes an event registration. Called when the App no longer wants cards to be read.

Request and Response TBD.

## Ticket Service API

## /ticket

Called by the Identity Agent only. Takes an Authentication Ticket and returns one or more Authorization Tickets.

Request

Response

## /id/create

Called by an enrollment app. Takes one or more App IDs with optional directed User IDs and returns the directed User ID for each App ID.

Request and Response TBD.

## /id/add

Called by an enrollment app. Takes two or more App IDs with directed User IDs and adds them to an existing User record.

Request and Response TBD.

## /id/get

Called by an enrollment app. Takes two or more App IDs and returns the directed User ID for each App ID.

Request and Response TBD.

## /id/delete

Called by an enrollment app. Takes one or more App IDs and deletes the associated entries in the User record.

Request and Response TBD.

## Data Provider API

The Data Provider may expose their API in any manner appropriate for the Data Provider. It is recommended the Data Provider expose a RESTful API and follow the parameter convention of:

app – App ID

ticket – Ticket for Data Provider

secret – The App’s secret for accessing the Data Provider

The Data Provider MUST confirm the value of the app parameter matches the value of app.gov.bc.ca in the Ticket.

# Security Considerations

* 1. Apps running locally in the browser cannot maintain secrets. These Apps MUST have a server component that makes authenticated calls to Data Providers to protect the DP\_Secret.
  2. The identity agent can remember AppID / ReturnURL combinations that were refused by the Ticket Service and fail immediately on those requests for a period of time, minimizing the impact of malicious apps using system and network resources.
  3. The interactions between the App and the Identity Agent can easily be intercepted and observed. The only mechanism for the Identity Agent to authenticate the App is to check if the Ticket Service generates an error that the Return URL was not registered to the App.
  4. TBC

# Privacy Considerations

* 1. User data MUST only be released by Data Providers to an App when the User has been presented with the data release description and authorized the transaction. Privileged Apps that have been verified by the Data Provider or a proxy and are acting in the best interests of the User or according to legislation may obtain User data without user authorization.
  2. The User identifiers used by the Ticket Service MUST only be used to correlate the User to other identifiers. They MUST NOT be exposed to any external process.
  3. The Authentication Service MUST NOT be presented with the App or Data Provider identifiers.

# Error Codes

Error codes assist Developers to determine the reason for failures while developing and deploying Apps, and enable the presentation of meaningful messages to assist Users in correcting the interaction.

USER\_CANCELLED

The User cancelled the transaction.

INVALID\_CREDENTIALS

Credentials provided by the User are not valid.

INVALID\_APP\_ID

The Ticket Service or a Data Provider did not recognize theAppID.

INVALID\_RETURN\_URL

The ReturnURL is not registered for thisAppID with the Ticket Service.

PASSCODE\_REQUIRED

The Data Provider requires the PassCode to be entered by the User for this transaction.

USER\_AUTHORIZAZ\_TicketION\_REQUIRED

The Data Provider requires the User to authorize the transaction.

UNKNOWN\_DAZ\_TicketA\_PROVIDER

The Ticket Service did not recognize the Data Provider ID.

INSUFFICIENT\_AUTHENTICAZ\_TicketION

The Data Provider requires stronger authentication than what was performed by the Identity Agent and/or Authentication Service.

# References

## Normative References

1. JSON Web Token (JWT) http://tools.ietf.org/html/draft-jones-json-web-token-10
2. JSON Web Encryption (JWE) <http://tools.ietf.org/html/draft-jones-json-web-encryption-02>
3. JSON
4. Language encodings http://tools.ietf.org/html/rfc5646

## Informative References

1. HTTP protocol
2. JSON
3. QR Codes
4. URI Schemes <http://www.iana.org/assignments/uri-schemes.html> <http://www.quora.com/Where-can-one-find-an-aggregate-list-of-all-protocol-handlers-that-all-apps-in-the-iTunes-App-Store-register-with-iOS>
5. X-callbadk-url 1.0 <http://x-callback-url.com/specifications/>