**High Level Design Document For**

**CSAPI v2.5**

|  |
| --- |
|  |
|  |

**Table of Contents**

[1. Title 5](#_Toc384112328)

[2. Introduction 5](#_Toc384112329)

[3. Purpose 6](#_Toc384112330)

[3.1. Target Audience 6](#_Toc384112331)

[3.2. Next Step 6](#_Toc384112332)

[4. Traceability 7](#_Toc384112333)

[4.1. Technical Mapping 7](#_Toc384112334)

[4.2. Functional Mapping 7](#_Toc384112335)

[4.3. Non-Functional Mapping 7](#_Toc384112336)

[5. Design Considerations 7](#_Toc384112337)

[5.1. Assumptions 7](#_Toc384112338)

[5.2. Limitations & Constraints 7](#_Toc384112339)

[6. System Architecture 7](#_Toc384112340)

[6.1. Logical Architecture 7](#_Toc384112341)

[6.2. Recommended Code Deployment 9](#_Toc384112342)

[6.2.1. Impact Analysis 10](#_Toc384112343)

[~~6.2.2.~~ ~~Context Diagram~~ 10](#_Toc384112344)

[6.3. User Interface 10](#_Toc384112345)

[6.4. Physical Architecture 10](#_Toc384112346)

[6.4.1. System Diagram 10](#_Toc384112347)

[6.4.2. Hardware elements 10](#_Toc384112348)

[6.4.3. Software elements 11](#_Toc384112349)

[7. Alternative Solutions and criteria for selection 11](#_Toc384112350)

[7.1. Design alternate 1 – Filter Status based on AppId (CSAPI v1.0) 11](#_Toc384112351)

[7.2. Design alternate 2 12](#_Toc384112352)

[7.3. Criteria for selection 12](#_Toc384112353)

[7.4. Solution selected with rationale 12](#_Toc384112354)

[8. System Details 12](#_Toc384112355)

[8.1. Input ID Validation 12](#_Toc384112356)

[8.2. Interrelationship and interfaces between the components 13](#_Toc384112357)

[9. Data Model 14](#_Toc384112358)

[9.1.1. Data Flow Table 14](#_Toc384112359)

[9.1.2. Database Design 16](#_Toc384112360)

[9.2. Application/System Integration - Request/Response Signatures 16](#_Toc384112361)

[9.2.1. CSAPI v1.0 Request Signature 16](#_Toc384112362)

[9.2.2. CSAPI v2.0 & later Request Signature 16](#_Toc384112363)

[9.2.3. CSAPI Response XML 20](#_Toc384112364)

[9.2.4. CSAPI Response Codes v1.0 & v2.0 26](#_Toc384112365)

[9.2.5. CSAPI Response Codes v2.5 26](#_Toc384112366)

[9.2.6. System Configuration Tokens 28](#_Toc384112367)

[10. Re-usable Components 29](#_Toc384112368)

[11. High level Security Design Assessment 29](#_Toc384112369)

[11.1. Sensitive Data 29](#_Toc384112370)

[11.2. Internal Interface 30](#_Toc384112371)

[11.3. External Interface 30](#_Toc384112372)

[12. Non Functional Requirements 30](#_Toc384112373)

[12.1. Automated Testing 30](#_Toc384112374)

[12.2. Performance Benchmarking 30](#_Toc384112375)

[12.3. Logging 31](#_Toc384112376)

[12.4. BI 31](#_Toc384112377)

[12.5. NOC Dashboard 31](#_Toc384112378)

[12.6. Availability 31](#_Toc384112379)

[12.7. Code Reusability – within CSAPI versions and across APIs 32](#_Toc384112380)

[13. Future Enhancements 32](#_Toc384112381)

[Revision History 33](#_Toc384112382)

# Title

Project Code: **PRJ201**

Project Name: **CSAPI v2.5**

System Name: **COMMON APIs**

Component Name: **COMMON APIs**

# Introduction

The initial Mind Tree version of common search API (hence forth called as version 1.0) has be written to lookup resident demographic/photo information based on EID/UID/ENO values. Mongo DB and Mongo Router are index on EID and is efficient only when the Mongo looks up are done based on EIDs.

However, Irrespective of the ID Type (be it EID, UID or ENO) being passed, the version 1.0 of Common Search API does SOLR looks up and fetched REFID and calls to MONGO to fetch the resident information.

This existing v1.0 workflow is inefficient for the following reasons:

1. REF ID is not indexed and Mongo is not efficient to look up based on REFIDs
2. SOLR indexing is manual process and there are gaps in existing SOLR indexing setup. All the lookups through SOLR may not return successful values which impacts the outcome of the Common Search API lookup
3. If EID is passed for Mongo Lookup, SOLR index look up should not happen and instead leverage the EID value to lookup Mongo directly.
4. V1.0 does not handle multiple id lookup gracefully and within the same request context
5. Using MySQL‘s UID Master and Origin Tracker tables, EID can be looked up when UID or ENO is passed. Leveraging this EID to look up Mongo will be more efficient and reliable compared to SOLR index lookup.

Finally, UID Master Photo is being sharded to multiple tables based on the first three digits of UID. Considering 0 and 1 values of UID first digit is reserved, 800 UID Master Photo tables are required to save all the resident photo information in MySQL.

CSAPI v2.5 is introduced to enrich the response with additional attributes like UID status, ref id based lookup, better error response codes to enable client application to make smart decisions.

Considering the photo sharding is complete in production, Common Search API v2.0 can leverage MySQL information to fetch resident demographic and photo information and only go to Mongo for the below scenarios:

1. When the calling application wants to lookup resident information where UID is not yet generated and therefore MySQL UID Master tables are not yet updated
2. Due to unknown technical reasons/exceptions MySQL do not have photo information available for a given resident.

# Purpose

# Target Audience

The relevance to this document is for the below teams, in the order precedence:

1. Architects
2. Developers
3. QA Team
4. Business Team

# Next Step

* Development team create Low Level Design Document (LLD)
* Develop the v2.5 service by re-factoring the existing code to make sure the code is modular and re-usable
* **The Low Level Design Document should create a detailed flow chart with various path based on request/response signatures, data flow and response codes defined in various sections of this document.**
* Implement all the non-functional features listed in this HLD

# Traceability

# Technical Mapping

<< BRD to Technical mapping >>

# Functional Mapping

<< BRD to functional mapping >>

# Non-Functional Mapping

# Design Considerations

## Assumptions

1. CSAPI v1.0 & v2.0 payload signatures (request/response) MUST not be changed
2. CSAPI v2.5 signatures have changed and requires client applications to be changed to take advantage of the new features.

## Limitations & Constraints

1. Nice to have features have been listed out in “Future Enhancement” section of this HLD.

# System Architecture

## Logical Architecture

Below is very high level code flow that explains the new workflow for v2.0 along with the existing v1.0 code base.

In summary, the existing v1.0 code is not changed except for handling SOLR search to return EID instead of REFID’s. This will ensure Mongo Lookups are faster for v1.0 calls as well.

Archival search is part of the existing CSAP v1.0 code base. This code is left as is to ensure backward compatibility with minimal change. Considering Most of the requests shall be handled in MySQL the probability of the production lookups coming to Mongo or Archival for scope=UIDLimited is very low.

***Version 2.1 does not do any additional data source lookups. However, it populated with additional information described in the Response XML (defined in below sections) so that client application can make informed decisions about the resident (packet).***

SearchRequestHandlerV2

***This module is changed to return EIDs instead of REFIDs***

If not found in MSQL, using eids, uids or enos

For uids or eids o enos (uidLimited)

scope=full && enos

For all uid, eid, enos, refids

New v2.0 call

Existing v1.0 call

Archival Search

MySQL Search

Mongo Search

SOLR Search

SearchRequestHandler

SearchResource

## Recommended Code Deployment

It is recommended that Status API shall be designed and developed as part of the same RPM deployment of CSAPI and ASAPI.

All the three APIs shall be developed in the same uidenrlsearchserver project scope. Maximum re-use of common validation classes and logic to be leveraged by this setup. This shall help improve the operational maintainability of the code in the long run.



## Impact Analysis

1. CSAPI service deployment should have no impact to the existing client applications using CSAPI.
2. CSAPI will always ensure true backward compatibility of its API versions with every new deployment of the service
3. It is the responsibility of each client application to migrate to the latest version to get additional features (as described in the API signatures) and take advantage of the newer API version.
4. Clients have to make code changes when migrating to CSAPI v2.5 as this latest version changes the response signature to provide additional information

## ~~Context Diagram~~

~~<<Very high level interaction between technical components of the system >>~~

## User Interface

***Not Applicable***

## Physical Architecture

To ensure high availability of the CSAPI service, the following deployment is highly recommended:

1. CSAPI service should be active in both the data center locations (BDC & GNDC)
2. CSAPI being read only service, always point to the read only replica’s of data bases – MySQL / Mongo available in each data center site
3. All the CSAPI instances should be hosted behind a software load balancer (equivalent to an apache reverse proxy) for high availability at each individual site

## System Diagram

***Not Applicable. Deployment recommendations are defined above***

## Hardware elements

* Deployed on the existing hardware of CSAPI
* 2 CPU, 4 Core machine
* At-least 16 GB of RAM

## Software elements

* Linux Operating systems
* Tomcat Application servers
* Apache Reverse Proxy (Software Load Balancer)
* MySQL Database servers
* BI Platform
* NOC Dashboard modules

# Alternative Solutions and criteria for selection

Considering UIDAI main business objective to service the residents/partners with identity, It becomes very important to ensure only the required information is shared to peripheral applications (e.g., resident portals, CRM, IVR, admin portals etc). This centralized service (aka CSAPI) will ensure sharing of resident’s PoI, PoA and related status information is secured and easily manageable to cater to various business requirements.

Any future changes with respect to schema related changes, sharding, securing of important columns from tampering or database technologies is all encapsulated by the CSAPI and all the ecosystem clients and implementation and roll out of these features are truly de-coupled.

# Design alternate 1 – Filter Status based on AppId (CSAPI v1.0)

The following are the possible statuses configured in uidv1\_1 schema status master table that will be allowed to be configured in properties file tokens and used by the appId filter:

|  |  |
| --- | --- |
| ***uid\_status\_key*** | ***uid\_status\_desc*** |
| 1 | Valid Aadhaar |
| 2 | Resident reported as Deceased |
| 3 | Aadhaar cancelled due to Duplicate |
| 4 | Aadhaar cancelled due to Fraud Enrollments |
| 5 | Aadhaar cancelled for Test Cases |
| 6 | Aadhaar cancelled due to Biometric Exception Cases |
| 7 | Aadhaar cancelled due to Cogent Flip Iris Cases |
| 8 | Aadhaar Suspended for Biometric Update |

1. ***If for any given application (appId), any of the above statuses are not configured then the CSAPI MUST not share this resident information out to the calling client application.***
2. ***Additionally introduction of scope = uidLimited or full, the CSAPI can intelligently cater to two broad set of user groups whose Aadhaar is generated or not respectively.***

# Design alternate 2

***Not Applicable***

# Criteria for selection

Both the above design requirements (functional / non-functional) are required. To ensure true backward compatibility of the CSAPI versions, the below table clearly defines features available in each of the CSAPI versions – v1.0, v2.0, v2.5.

# Solution selected with rationale

***Not Applicable***

# System Details

## Input ID Validation

The following validations to be done for each of the input id types:

1. ***UID***
   1. Should be 12 digit base10 numeric number
   2. Should pass the verhoeff checksum logic on the 12th digit
2. ***ENO***
   1. Remove “/” signs if any in the input string
   2. Should be 14 digit base10 numeric number
   3. First 4 digits has to map to a Enrolment Agency Id from master table
   4. 5th to 9th digit should map to a station id from master table
   5. 10th to 14th digit cannot be validated and will be valid number
3. ***EID***
   1. Remove “/” signs if any in the input string
   2. Should be 28 digit base10 numeric number
   3. First 14 digits should be validated as per the above ENO validation steps
   4. The remaining 14 digits should be a valid YYYYMMDDhhmmss data format
   5. The date should be greater than Jan 1st 2010 000000 hrs
4. ***REF ID***
   1. Should match to this regular expression of a typical Java UUID string:

[0-9a-fA-F]{8}(?:-[0-9a-fA-F]{4}){3}-[0-9a-fA-F]{12}

* 1. i.e alpha numeric chars of length 8, 4, 4, 4 and 12
  2. each set separated by a “-“ digit
  3. Total length comes to 36 characters in length.

If any of the above validation steps fail, raise the appropriate error code (refer to the Response Code section of this HLD document for detailed error codes).

*Low Level Design must explore any available java utilities that do the ref id based validation instead of writing net new code. Verhoeff checksum logic is also available in platform code base and should be used instead of writing new code.*

## Interrelationship and interfaces between the components

***Not Applicable***

# 

# Data Model

## Data Flow Table

| **Input** | **Scope** | **Status Filter** | **v2.0 Flow/Hits (v2.5 specific highlighted in grey)** | **Remarks** |
| --- | --- | --- | --- | --- |
| UID | UIDLimited | App Id passed | * UID master to get demo/photo/uidStatus * If NO PHOTO,   + Get EID from origin tracker,   + and then get latest photo from Mongo | * If AppId is not passed, assumes default configured statuses. * The enhanced FI code base will use this option to fetch multiple photos (batch of 200) in one rest call. Will leverage staging testing to get additional inputs to measure the performance of this call. * ***Origin Tracker can have multiple records against an UID. Pick the latest one with photo updated from Mongo.*** |
| <null or full or any other value> |  | * Using passed UIDs. Get corresponding EID from origin tracker & uidStatus from uid master * Search Mongo with EIDs * *If UID is available get uidStatus (v2.5 and later)* | * Statuses filter is not applied |
| EID | UID Limited | App Id Passed | * My SQL UID origin tracker to get UID * UID master to get demo/photo/uidStatus * If NO PHOTO available in MySQL   + Get Mongo using passed EID | * If AppId is not passed, assumes default configured statuses. |
| <null or full or any other value> |  | * Mongo using passed EID * ~~Refresh Mongo entity from MySQL using Mongo.UID~~ * Get UID Status for those EID who have an UID | * Statuses filter is not applied |
| ENO | UIDLimited | App Id Passed | * MySQL UID origin tracker to get UID Master Entity * UID master to get demo/photo/uidStatus * If NO PHOTO available in MySQL   + Get EID from origin tracker,   + Get Mongo using looked-up EID | * If AppId is not passed, assumes default configured statuses. * ***Origin Tracker can have multiple records against an UID. Pick the latest one with photo updated from Mongo.*** |
| <null or full or any other value> |  | * SOLR  get EID * Mongo using EID from SOLR * ~~Refresh Mongo entity from MySQL~~ * Get UID Status for those EID who have an UID | * Statuses filter is not applied |
| REFID | UIDLimited |  | Not Supported | Not Supported |
| <null or full or any other value> |  | * SOLR  get EID * Mongo using EID from SOLR * ~~Refresh Mongo entity from MySQL~~ * Get UID Status for those EID who have an UID | * Statuses filter is not applied |

## Database Design

***Not Applicable***

# Application/System Integration - Request/Response Signatures

# CSAPI v1.0 Request Signature

The Common Search API v2.0 is designed to completely co-exist with version 1.0. The only change done to the existing version 1.0 is when SOLR Index is looked up EID is returned instead of REFID. This is minor change and will improve the Mongo lookup more efficiently.

# CSAPI v2.0 Request Signature

The Rest call is differentiated between versions from the existence of version tag (e.g **2.0, 2.1 etc)** in the URI:

***Example URI***: **http://<commonsearchapi\_server\_ip>:8080/enrlsearch/2.0?scope=uidLimited&appId=EKYC"**

# CSAPI v2.5 Request Signature

CSAPI v2.5 has been enhanced to get all the attributes into request xml payload. in the URI:

***Example URI***: **http://<commonsearchapi\_server\_ip>:8080/commonsearch/2.5/**

**<xs:import namespace=**[**http://www.uidai.gov.in/searchapi/common/types/2.5**](http://www.uidai.gov.in/searchapi/common/types/2.5)

**schemaLocation="searchapi-common-types-v2.5.xsd" />**

**<xs:element name="CommonSearch">**

**<xs:complexType >**

**<xs:sequence>**

**<xs:attribute name="appID" type="common:AppID" use="required"/>**

**<xs:attribute name="txnID" type="commmon:Txn" use="required"/>**

**<xs:attribute name="userID" type="common:UserID" use="optional"/>**

**<xs:attribute name="ver" type="common:Ver" use="required"/>**

**<xs:attribute name="lk" type="common:LicenseKey" use="optional"/>**

**<xs:attribute name="ts" type="xs:string" use="required"/>**

**<xs:attribute name="scope" type="common:SearchScope" use="required"/>**

**<xs:attribute name="filter" type="DataFilter" use="required"/>**

**<xs:attribute name="idType" type="RequestIDType" use="required"/>**

**<xs:element name="id" type="xs:string" minOccurs="1" maxOccurs="unbounded" />**

**</xs:sequence>**

**</xs:complexType>**

**</xs:element>**

| **Option** | **Supported API Version** | **Default Value** | **Description** |
| --- | --- | --- | --- |
| enrlsearch | 1.0 & 2.0 |  | Rest endpoint uri |
| commonsearch | 2.5+ |  | Rest endpoint uri |
| appID | 2.0 + | DEFAULT | If **appID** request attribute is present in the request then API will consider only those resident’s data to be returned back, whose current status matches to one of the configured statuses of that application id. If this attribute is not present then the default statuses are assumed. This option acts as a security filter to **NOT** return resident information for which application is not allowed to cater to. |
| txnID | 2.5+ |  | 50 alphanumeric char (with ‘-‘) set by calling client to identify a transaction. Can be set as UUID value format. |
| userID | 2.5+ |  | **Optional**  Max 12 character userId value for auditing.  Application like CRM can decide to provide this information for auditing. |
| ver | 2.5+ |  | This part of rest URI and payload as well. Determines the rest end point version to call. |
| lk | 2.5+ |  | **Optional** attribute. This will be used for future use to control api usage by calling client application. |
| ts | 2.5+ |  | Timestamp set by the calling client |
| Scope | 2.0 + | full | If ***scope value*** is present in the requestand contains ‘***uidLimited***’ value, then API will first lookup in MySQL and if not successful then look up in Mongo/Archival. |
| filter | 1.0+ |  | DEMOGRAPHICS  DEMOGRAPHICS\_PHOTO |
| IDParameterField | 1.0 & 2.0 only |  | uid, eid, eno, \_id (for ref id is only in 2.0) |
| idType | 2.5+ |  | uid, eid, eno, \_id. \_id refers to ref id.  ***REF ID lookup is highly NOT recommended for lookup against Mongo data source using scope=full for performance reasons.*** |
| id | 2.5+ |  | Max 200 request Id are allowed in the id tag |
| Filter | 1.0 + |  | DEMOGRAPHICS  DEMOGRAPHICS\_PHOTO |
| maxRecordsToRead | 1. &2.0 | 10 | Max records to lookup |

**Additional Notes (IMPORTANT):**

1. Common Search API v2.0 and later is enabled to gracefully handle multiple ids within the same request and provide the aggregated response to the calling clients. Though the existing version 1.0 API request/response is designed to handle this but the code was not handling this use case.
2. The v2.0 API handles multiple ids gracefully and attempts to make best effort to fetch resident information. To use this feature calling applications DO NOT require changing the existing request/response xsd’s.
3. To keep the new v2.0 workflow simple, there is a hard check to ensure the same request does not contain multiple id types. i.e calling client application can not send a mix of eid’s, uid’s and eno’s in the same request. In other words id type is mutually exclusive with in a context of a request.
4. In v2.5, the request signature is changed to send only one type of idType for all the requestId’s in the request. This idType MUST be one of be a valid supported value.
5. In v2.5, the response codes are enhanced to be more specific both at the request level and requestId levels of response XSD.

# CSAPI Response XML

The below table describes clearly the difference between the v1 and v2 response xml. Detailed response elements supported in each version of API options is listed below as Y or N.

However when scope=full option is used; CSAPI v2.0 supports all the tag nodes.

| **Parent Tag** | **Tag** | **Comments** | **v1.0 &**  **v2.0 with scope=full** | **v2.0 with scope=uidLimited** | **v2.5** |
| --- | --- | --- | --- | --- | --- |
| *status* |  | v1.0 & 2.0:  *0 = failed; 1 = success.*  *v2.5: detailed statuses are available* | Y | Y | N |
| *CommonSearchResp* | appID (attribute) |  | N | N | Y |
| *CommonSearchResp* | txnID (attribute) |  | N | N | Y |
| *CommonSearchResp* | result (attribute) | y or n | N | N | Y |
| *CommonSearchResp* | err (attribute) | Only if result = n | N | N | Y |
| *CommonSearchResp* | respCode (attribute) | New UUID for each request | N | N | Y |
| *CommonSearchResp* | ts (attribute) | Server response time stamp | N | N | Y |
| *record* | idType (attribute) | *Requested idType uid,eid,eno or refid AS IS(helps map the request with response by the clients)* | N | N | Y |
| *record* | id (attribute) | *Requested id* | N | N | Y |
| *record* | result (attribute) | *y or n* | N | N | Y |
| *record* | err (attribute) | *Only if result = . e,.g 201,202,300,500 etc* | N | N | Y |
| *record* | uidStatus (attribute) | *This value Must be always be populated even if uidStatus fails due to appId filter test.* ***If UID Is not available then uidStatus = -1*** | N | N | Y |
| *record.packetStatus* | attribute |  |  |  |  |
| *Record.* *uidIssuedOrRejectedDate* | attribute |  |  |  |  |
| *record.demographics* |  |  | Y | Y | Y |
| *record.demographics* | *refId* | *This value shall be populated in response only if the request is based on refId and with scope=full.* | Y\* | N | Y\*\* |
| *record.demographics* | *Eid* |  | Y | Y | Y |
| *record.demographics* | *Name* |  | Y | Y | Y |
| *record.demographics* | *enrolment-number* |  | Y | Y | Y |
| *record.demographics* | *enrolment-date* |  | Y | Y | Y |
| *record.demographics* | *enrolmentType* | *{NEW, UPDATE, CORRECTION, QACORRECTION, LOSTUID}. Populated only if scope=full* | N *(only for DDCV1 appId as exception)* | N *(only for DDCV1 appId as exception)* | Y |
| *record.demographics* | *Gender* |  | Y | Y | Y |
| *record.demographics* | *Dob* |  | Y | Y | Y |
| *record.demographics* | *dob-status* |  | Y | Y | Y |
| *record.demographics* | *Pincode* |  | Y | Y | Y |
| *record.demographics* | *po-name* |  | Y | Y | Y |
| *record.demographics* | *po-type* |  | Y | N | Y\*\* |
| *record.demographics* | *Careof* |  | Y | Y | Y |
| *record.demographics* | *Building* |  | Y | Y | Y |
| *record.demographics* | *Street* |  | Y | Y | Y |
| *record.demographics* | *Landmark* |  | Y | Y | Y |
| *record.demographics* | *Locality* |  | Y | Y | Y |
| *record.demographics* | *Vtc* |  | Y | Y | Y |
| *record.demographics* | *vtc-name* |  | Y | Y | Y |
| *record.demographics* | *District* |  | Y | Y | Y |
| *record.demographics* | *district-name* |  | Y | Y | Y |
| *record.demographics* | *sub-district* |  | Y | Y | Y |
| *record.demographics* | *sub-district-name* |  | Y | Y | Y |
| *record.demographics* | *sub-district-local-name* |  | Y | Y | Y |
| *record.demographics* | *State* |  | Y | Y | Y |
| *record.demographics* | *state-name* |  | Y | Y | Y |
| *record.demographics* | *Mobile* |  | Y | Y | Y |
| *record.demographics* | *Email* |  | Y | Y | Y |
| *record.demographics* | *lang-code* |  | Y | Y | Y |
| *record.demographics* | *local-res-name* |  | Y | Y | Y |
| *record.demographics* | *local-careof* |  | Y | Y | Y |
| *record.demographics* | *local-building* |  | Y | Y | Y |
| *record.demographics* | *local-street* |  | Y | Y | Y |
| *record.demographics* | *local-landmark* |  | Y | Y | Y |
| *record.demographics* | *local-locality* |  | Y | Y | Y |
| *record.demographics* | *local-vtc* |  | Y | Y | Y |
| *record.demographics* | *local-district* |  | Y | Y | Y |
| *record.demographics* | *local-state* |  | Y | Y | Y |
| *record.demographics* | *po-name-local* |  | Y | Y | Y |
| *record.demographics* | *information-sharing-consent* |  | Y | Y | Y |
| *record.demographics* | *Uid* |  | Y | Y | Y |
| *record.demographics* | *residentPhoto* |  | Y | Y | Y |
| *record.demographics* | *exceptionPhoto* |  | Y | N | Y\*\* |
| *record.demographics* | *isCurrent* |  | Y | Y | Y\*\* |
| *record.demographics* | *updatedRefIds* |  | Y | N | Y\*\* |
| *record.demographics.relation-info* |  |  | Y | N | Y\*\* |
| *record.demographics.relation-info* | *rel-type* |  | Y | N | Y\*\* |
| *record.demographics.relation-info* | *rel-name* |  | Y | N | Y\*\* |
| *record.demographics.relation-info* | *rel-local-name* |  | Y | N | Y\*\* |
| *record.demographics.relation-info* | *rel-id* |  | Y | N | Y\*\* |
| *record.demographics.relation-info* | *rel-id-type* |  | Y | N | Y\*\* |
| *record.demographics.processingInfo* |  |  | Y | Y | N |
| *record.demographics.processingInfo.field* |  |  | Y | Y | N |
| *record.demographics.processingInfo.field* | *uidIssuedOrRejectedDate* |  | Y | Y | N |
| *record.demographics.processingInfo.field* | *stageStatus* |  | Y | N | N |
| *record.demographics.processingInfo.field* | *packetState* |  | Y | N | N |
| *record.demographics.processingInfo.field* | *reasonCode* |  | Y | N | N |
| *record.demographics.processingInfo.field* | *rejectReasonCode* |  | Y | N | N |
| *record.demographics.processingInfo.field* | *REJECTED* |  | Y | N | N |
| *record.demographics.processingInfo.field* | *COMPLETED* |  | Y | Y | N |
| *record.demographics.processingInfo.value* | Value | holds the field value | Y | Y | N |
| *record.auditInfo* | operatorName |  | N | N | Y\*\* |
| *record.auditInfo* | supervisorName |  | N | N | Y\*\* |
| *record.auditInfo* | introducerName |  | N | N | Y\*\* |
| *record.auditInfo* | registrarName |  | N | N | Y\*\* |
| *record.auditInfo* | agencyName |  | N | N | Y\*\* |

***\* Only with v2.0, scope=full and Request ID Type = Ref ID***

***\*\* Only when scope=full***

|  |  |  |
| --- | --- | --- |
| XSD Type | Version | XSD File |
| common-search-request-v2.5 | 2.5 |  |
| common-search-response-v2.5 | 2.5 |  |
| searchapi-common-types-v2.5 | 2.5 |  |

# CSAPI Response Codes v1.0 & v2.0

To ensure 100% backward compatibility with all the calling client application, the request payload and response signatures are kept intact between version 1.0 & v2.0.

However CSAPI v2.0 added few additional request parameters that do not warrant for a client application code change to bring in efficiency of the lookup logic. This has been already defined in detail above.

# CSAPI Response Error Codes v2.5

CSAPI v2.0 introduces two levels of responses.

* + 1. At the request level, the response code caters to validation of all the mandatory request attributes and service is available to process the request###:

|  |  |  |
| --- | --- | --- |
| err Code | Reason | Comments  (for internal developer understanding/guidance) |
| 201 | No Records Found | Check the idType or idValues again and try |
| 202 | Too Many Records Found | More than one record found when searching against the ID. For e.g, ENO results more than one record. EID or REF ID results in more than one record in Mongo! Please contact support team. |
| 300 | Invalid Request | Request is invalid, Please check the request signature/options and try again. |
| 301 | Invalid txnId | txnId is invalid. Please check and try again |
| 302 | Invalid requestIdType | Check the requestIdType value and try again |
| 303 | Invalid requestId | requestId is failed with basic validation checks. Please check the requestId and try again. |
| 304 | Invalid filter | Filter value is not valid. Please check and try again |
| 305 | Invalid maxRecordsToRead | maxRecordsToRead is not valid. Please check and try again. |
| 306 | Invalid userId | userId is invalid. Check the length and try again. |
| 500 | Technical Exception | Check the logs for detailed technical errors |
| 501 | Internal Data Error | One or more of the mandatory database columns are not available to process the request. Please contact support team. |

* + 1. For each individual request id, a detailed response code shall provide additional intelligence to the calling applications (especially the batch jobs that call multiple id look ups)###:

|  |  |  |
| --- | --- | --- |
| err Code | Reason | Comments  (for internal developer understanding/guidance) |
| 201 | No Records Found | Check the id again and try |
| 202 | Too Many Records Found | More than one record found when searching against the ID. For e.g, ENO results more than one record. EID or REF ID results in more than one record in Mongo! Please contact support team. |
| 203 | Record filtered | Resident record is filtered out due to appId is not allowed to cater to this uidStatus. **NOTE: uidStatus attribute MUST be populated. If not UID found set uidStatus to -1** |
| 350 | Invalid Id | requestId is invalid. **NOTE: uidStatus attribute must be set to -1** |
| 500 | Technical Exception | Check the logs for detailed technical errors |
| 501 | Internal Data Error | One or more of the mandatory database columns are not available to process the request. Please contact support team. |

***###If request level response is of 3XX or 5XX series errors, then individual records need not be populated in the response xml.***

***The Low Level Design Document should create a detailed flow chart with various path based on this document request/response signature definitions.***

# System Configuration Tokens

For Common Search API v2.0 and later to consider a new application id, there is no code changes required but expand the ***COMMONSEARCH.API.RESIDENT.UID.CLIENT.LIST*** token with the new appId and then create a new token with the above format for the new appId.

# UIDMASTER.SHARDING.ENABLED is used to get the shard hint based on UID/EID/REFID while interacting with sharded uid\_masterv1\_1 schema

**UIDMASTER.SHARDING.ENABLED=false**

#CLIENT.LIST helps define the list of client’s apps (ALL IN UPPERCASE) using COMMON SEARCH API services.

#Each of the below APPID MUST HAVE statuses TOKENS defined.

**COMMONSEARCH.API.RESIDENT.UID.CLIENT.LIST=DEFAULT,EKYC,EAADHAAR,PORTAL,FI,MDD**

#STATUSES.FOR.<***APPID***> is used to define the active statuses list that commons search API would filter out to process the requests.

**COMMONSEARCH.API.RESIDENT.UID.STATUSES.FOR.DEFAULT=1**

**COMMONSEARCH.API.RESIDENT.UID.STATUSES.FOR.EKYC=1**

**COMMONSEARCH.API.RESIDENT.UID.STATUSES.FOR.EAADHAAR=1**

**COMMONSEARCH.API.RESIDENT.UID.STATUSES.FOR.PORTAL=1**

**COMMONSEARCH.API.RESIDENT.UID.STATUSES.FOR.MDD=1,2,3,4,5,6,7,8**

# Re-usable Components

As number of client applications using Status API grows, it becomes a challenge to expect all the clients to adapt to a new version on day-1.

It becomes mandatory for Status API to support multiple versions at any given point in time, so that the clients are not impacted. This begs a question how to structure the code in Status API so that maximum code re-use is done without duplicating common functionality and logic across various versions.

Starting from version 1.0, the code should be structured to have an abstract base class that holds common code and any unique logic that varies between versions will only be implemented in the respective version handlers.

The rest end point must implement either using reflections or delegate patterns to invoke the right version of the API.

Finally, CSAPI (Common Search API), ASAPI (Advance Search API) and STAPI (Status API) shall all be hosted together in the same ***enrlsearchserver*** rpm. Every effort has to be made to call out the common utility and helper classes to the common package that can be re-used across all the three APIs.

*For e.g Validation of EIDs, ENOs, UID and REFID logic can be common re-usable method.*

# High level Security Design Assessment

## Sensitive Data

Considering the sensitivity of the information processed by Status API, the service shall be accessible only within the CIDR premises and is not directly exposed to the outside clients. The service is hosted behind the reverse proxy in DMZ location.

**IP Address from the http request header must always be logged across:**

* + 1. **Application Service log file**
    2. **NOC**
    3. **BI events**

**This will enable additional real time and offline monitoring of API usage**

## Internal Interface

***This component is available only for internal applications.***

## External Interface

***Not Applicable***

# Non Functional Requirements

# Automated Testing

Considering the Common Search API is restful end point with same xsd contract between versions (v1 and v2) by the calling clients and the service, this becomes a perfect candidate to automate the testing to validate the v2 change by comparing field to field values.

The automated tool shall be developed with the following objectives:

1. Easily configure the CSAPI environment to point to for comparing the results
2. Read the list of IDs (eid or uid or eno) from an input csv file. This list shall be used to fetch CSAPI response against v1 and v2 rest endpoints
3. Configure a token that shall easily allow to filter out those tags with known differences and ignore in the comparison
4. For each id where there is difference detected, generate a report with each of the attribute values between the versions
5. Tool shall be able to handle huge volumes as part of the input CSV file and easily generate the report

# Performance Benchmarking

CSAPI is designed to be stateless and should handle at least 30 M requests over a 10 hour window.

The new version capability to lookup into local MySQL tables should help easily lookup data without any additional hops done in v1 - like the SOLR/Mongo. The upgraded version of Mongo shall also provide read affinity between the BDC and GNDC locations and should make the CSAPI service requests out of the local data sources.

Benchmarking tests shall be done to plan the infrastructure that can handle at least 30M over a 10 hour window with sub second responses.

# Logging

CSAPI v2.0 shall handle all the exception scenarios more gracefully and ensure adequate information is available with the appropriate log level.

To ensure resident privacy, No personal information related to resident should be logged. EnrlSearchException handler should be leveraged to pass on the exceptions with appropriate messages bubble out the restful end point to log the appropriate message. This shall help operations for easy troubleshooting.

# BI

BI Event has to be published with the following elements:

1. *txnID (ID to identify a transaction. This value is set by calling clients)*
2. *appId*
3. *userId*
4. *ts*
5. ***ip-address (from the http request header)***
6. *respCode*
7. *result (y or n)*
8. *err*
9. *scope*
10. *idType*
11. *id* ***(trimmed and SHA1 Hashed if EID/ENO/UID)***
12. *id-result (individual request id result)*
13. *id-err (individual request id err)*
14. *request-ts (*YYYY-MM-DDThh:mm:ss)
15. *response-ts (*YYYY-MM-DDThh:mm:ss)
16. *response time(at request level aggregating all request Ids together in ms)*

# NOC Dashboard

NOC event shall be published to the NOC rest end point.

However, consumption of this information is not in scope of this project.

# Availability

CSAPI is highly available service and is hosted between BDC & GNDC locations. Each location hosts multiple CSAPI instances behind a load balancer and reverse proxy.

# Code Reusability – within CSAPI versions and across APIs

As number of client applications using CSAPI grows, it becomes a challenge to expect all the clients to adapt to a new version on day-1.

It becomes mandatory for CSAPI to support multiple versions at any given point in time, so that the clients are not impacted. This begs a question how to structure the code in CSAPI so that maximum code re-use is done without duplicating common functionality and logic across various versions.

At least starting version 2.0 and 2.1, the code should be structured to have an abstract base class that holds common code and any unique logic that varies between versions will only be implemented in the respective version handlers.

Finally, CSAPI (Common Search API), ASAPI (Advance Search API) and STAPI (Status API) shall all be hosted together in the same ***enrlsearchserver*** rpm. Every effort has to be made to call out the common utility and helper classes to the common package that can be re-used across all the three APIs. *For e.g Validation of EIDs logic can be common re-usable method.*

# Future Enhancements

* + When multiple ids are passed within a request, the current MySQL lookup does one id lookup at a time. This can be enhanced to fetch all the id’s in one go from MySQL.

Considering all the existing clients call with only one ID, this enhancement can be taken up after the uid master is sharded across 80 schemas.

* + ~~[~~**~~Not Applicable~~**~~] If resident demographic is available in MySQL but had to goto Mongo to fetch Photo, the v2.0 workflow can be enhanced to automatically to update the MySQL with the photo. This will ensure any subsequent lookup for that resident is directly fetched from MySQL.~~
  + Enable enumeration for while configuring valid statuses that are applicable for an application. Currently it is listed as status id integer values defined in uid master status table.
  + ~~[~~**~~Not Applicable~~**~~] Enhance to set the exception photo read from MySQL photo tables. Exception Photos are not required in scope=uidLimited~~
  + Enhance the validation of IDs passed in the request. For e.g UID should be a 12 digit number where first digit cannot be 0 or 1. EIDs should adhere to the standard EID format with valid EA/Station Id codes and valid date format. The detailed steps in included above in v2.5 HLD.
  + Relationship information should be populated only in the full scope. Also the relationship UID alone needs to be passed AND NOT EIDs. Relationship UIDs should be in the valid status as configured for each of the applications. Similarly updatedRefID list to be passed out only in the full scope.

# Revision History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **VERSION NO.** | **DATE** | **AUTHOR** | **Description for Change** | **REVIEWED BY** | **APPROVED BY** |
| 1.0 | 30-08-2013 | Vijay Vujjini | First draft |  |  |
| 1.01 | 02-09-2013 | Vijay Vujjini | Enhanced ENo based lookup considering scope request parameter. |  |  |
| 1.02 | 13-09-2013 | Vijay Vujjini | Looking up latest Enrolment record for Photo from Mongo |  | Pramod/Vivek |
| 1.03 | 23-09-2013 | Vijay Vujjini | Feeback from Pramod/Vivek Review – Detailed prod status reviewed from each app context. ENO lookup is more streamlined to enable UIDLimited scope as well. Added NFR section. |  |  |
| 1.04 | 17-10-2013 | Vijay Vujjini | Non-critical Issues noticed from automated run tracked in future enhancements |  |  |
| 1.05 | 31-10-2013 | Vijay Vujjini | Difference between v1 and v2 xml response tags |  |  |
| 1.06 | 29-11-2013 | Vijay Vujjini | Ref Id based lookup support for MDD. REFID is returned in response as well. NOC logging… |  |  |
| 1.08 | 04-03-2014 | Vijay Vujjini | Support passing uidStatus, enrolmentType, refId in the response as a minor CSAPI version enhancement v2.5. Added few additional Non-Functional requirements |  |  |
| 1.09 | 28-03-2014 | Vijay Vujjini | New HLD Template Format & Detailed response codes for CSAPI v2.5 | Vivek/Pramod |  |
| 1.10 | 10-04-2014 | Vijay Vujjini | Xsd’s should be uniform in terms of attribute naming conventions and include all standard attributes. | Vivek | Vivek |
| 1.11 | 10-04-2014 | Vijay Vujjini | Minor updates to ts format | Vivek | Vivek |
| 1.12 | 29-05-2014 | Vijay Vujjini | Po-type is not in uid address table. This tag will be available only in scope=full. Minor correction on enrolment-type tag presence across api versions. |  |  |