Contents

• Welcome to Indivo X’s technical documentation!
  – Overview
  – General Interest
  – App Developers
  – Indivo Administrators
  – Indivo Hackers
  – HOWTOs
  – Release Notes
• Indices and tables
Overview

If you are interested in working with Indivo, you probably fall into one or more of the following categories:

- **General Interest**: You’ve heard about this ‘Indivo X’ project, and want to learn more about what it can do. You might want to check out our non-technical site before diving into this more technical overview.

- **App Developers**: You want to build applications that enable patients to engage in their care, using Indivo’s RESTful API to store and fetch medical data. You are interested in the API, including functionality and authentication.

- **Indivo Administrators**: You want to set up and host your own instance of Indivo, so that you or others can build applications on top of it. You are interested in pulling data into Indivo, setting up users, and managing your instance.

- **Indivo Hackers**: You are a big fan of Indivo, but realize (as do we) that there are plenty of additions that could make Indivo X even better! You want to dive into Indivo’s core codebase and fix bugs or add features, and contribute back to our open-source community.

Based on your level of interest, check out the documentation below.
CHAPTER 2

General Interest

2.1 Indivo API

The Indivo API provides an interface for Personal Health Applications to extend the functionality of the Indivo PCHR. The Indivo API abides by the REST design pattern: it is stateless, re-uses existing HTTP constructs such as caching, compression and content negotiation, and generally uses URLs to represent hierarchical resources, e.g. documents within a medical record.

This document provides an introduction to the API, with in-depth explanations of related core Indivo concepts where applicable. For a full listing of all available Indivo API calls, see API Reference.

2.1.1 Overview

Personal Health Applications (PHAs) make HTTP calls to the Indivo API endpoint using the REST convention. oAuth is used to authenticate all calls, either in 2-legged mode for simple authentication, or in 3-legged mode when the PHA is making a call with delegated authentication, i.e. on behalf of the user.

Application Types

We consider three types of applications:

- **User Applications**, which individual Indivo users can add to their record.
- **Administrative Applications**, which are used to perform account and record manipulations.
- **UI Applications**, which provide the public user interface to Indivo features.

Per Indivo installation, there is a small handful of UI and administrative applications, and quite a number of user applications.

Terminology

A **record** is the single set of medical information that pertains to an individual. It is composed of **documents**, including a **demographics document** which details the individual’s contact information and name. A record can be accessed by one or more **accounts**.

**oAuth** is the authentication protocol used by Indivo. In 2-legged oAuth, the PHA (the oAuth **consumer**) makes calls to Indivo (the oAuth **service provider**) using a **consumer key** to identify itself, and a **consumer secret** to sign the request. In 3-legged oAuth, the PHA makes calls to Indivo to access medical information as delegated by the user, using an additional **token** and **token secret** that pertain to the specific Indivo record being accessed.
2.1.2 Authentication

All calls to Indivo are authenticated using oAuth.
We detail the authentication process at Indivo Authentication.

2.1.3 Design Patterns

Some common design patterns are repeated throughout the API. Not all of these patterns are necessarily 100% supported by all Indivo API implementations, though when they are they are consistent.

Email Addresses as Identifiers

Core accounts in Indivo X are identified by email addresses, because email addresses provide mechanisms for distributed identification and messaging. When an email address is included in a URL, it must be URL encoded, where the @ sign turns into %40.

Paging/Filtering Results

When a list of results are returned, the URL ends in a / and the HTTP method is a GET, as is typical of REST design. In that case, Indivo X supports a generic query string that determines paging and ordering of the results:

?offset={offset}&limit={limit}&order_by={order_by}&status={document_status}&modified_since={modified_since}

- offset indicates which item number to start with, e.g. when getting a second batch of items.
- limit indicates the maximum number of items to return. This is used in combination with offset to accomplish paging.
- order_by is dependent on the fields returned in the list of items, and each call must thus define which fields are valid. Using an invalid field in order_by results in no effect on the output, as if order_by were absent.
- status can be used where applicable. It pertains to the status of documents and can currently be set to one of three options: ‘void’, ‘archived’ or ‘active’
- modified_since allows an application to look at items that have been modified since a given timestamp, so that incremental downloads may be possible.

Querying Results

As of the Beta3 release, calls that implement the basic paging operations above may also implement a more powerful query interface, also represented in the query string. In these cases (currently all of the minimal medical reports and the auditing calls), the following values may occur in the query string:

?qoffset={offset}&limit={limit}&order_by={order_by}&status={document_status}

These values function as before.

?group_by={group_field}&aggregate_by={aggregation_operator}*{aggregation_field}

group_by groups results by the specified field. It must be used in conjunction with aggregate_by, which aggregates the results by group, using the specified operation. If aggregate_by is passed without a group_by parameter, the aggregation is performed over the entire result set. Results that have been aggregated are returned in an aggregated format, not the typical reporting format.
?date_range={date_field}*{start_date}*{end_date}

date_range filters results and leaves only those with the specified field falling between start_date and end_date.

?date_group={date_field}*{time_increment}&?aggregate_by={aggregation_operator}*{aggregation_field}

date_group functions equivalently to group_by, except the groups are formed based on the values of the specified date field. For example, if the date field was ‘date_measured’, and the time increment was ‘month’, results would be returned grouped by the month of the date_measured field for each item. As with group_by, date_group must be used with an aggregator, and results are returned in an aggregated format.

?{FIELD}={VALUE}

This syntax adds additional filters to the query, returning only results having whose value for the property specified by ‘field’ matches ‘value’.

For each of these parameters, acceptable values for {field} are specified individually by the calls. A full listing of the minimal reporting fields, along with valid aggregation operators and date increments, may be found here.

External IDs

When a resource is created, the Indivo API offers the ability to create this resource using a PUT with an external_id in the URL, so that the call is idempotent: if a failure occurs, the call can be repeated safely and only the resource will not be created on the second call if it was already created successfully during the first call.

An external_id is only valid within a particular PHA scope. Other PHAs cannot see the external_id of a given document if they didn’t create the document, and certainly cannot access the document by external_id.

Some API calls which involve both creating documents and retrieving them, such as:

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/external/{APP_ID}/{EXTERNAL_ID}/

For these calls, it can be confusing as to which document is referenced by an external id. In such cases, the following rule resolves confusion:

• The newly created document will always be assigned the passed external_id. The external_id will not
be used to look up the existing document.

2.1.4 Managing Documents

Data stored in Indivo cannot by permanently deleted by default: the API enforces only appending data, not fully replacing it or removing it.

Reading Documents

GET /records/{RECORD_ID}/documents/, GET /carenets/{CARENET_ID}/documents/

List documents within a record. Supports order by document metadata fields (see Indivo Document Metadata Schema).

The calls to GET /records/{RECORD_ID}/documents/ and GET /carenets/{CARENET_ID}/documents/ take a type querystring parameter, which filters the list of returned documents by their types.

A document’s type is (by default) the suffix of a URL that corresponds to the XML schema datatype, where the prefix is http://indivo.org/vocab/xml/documents#. Thus, type can be Medication, Lab, etc.
Indivo X supports storing XML documents whose datatype is not among the default Indivo X recommended types. In those cases, if the XML schema namespace doesn’t end in a / or #, then as is typical in the XML/RDF community, a # is used as delimiter in the URI. Examples of document types include:

- http://indivo.org/xml/phr/medication#Medication (Indivo 3.1 data type)
- urn:astm-org:CCR#ContinuityOfCareRecord, as per http://code.google.com/apis/health/ccrg_reference.html

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}, GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID}  
Fetch a single document.

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/meta, GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID}/meta  
Fetch metadata about a single document, using its internal or external id.

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/versions/  
List versions of a single document.

Writing Documents

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/label, PUT /records/{RECORD_ID}/documents/external/{APP_ID}/{EXTERNAL_ID}/label  
Update a single document’s label.

POST /records/{RECORD_ID}/documents/, PUT /records/{RECORD_ID}/documents/external/{APP_ID}/{EXTERNAL_ID}  
Create a new document, and possibly assign it an external id.

Medical data cannot be replaced wholesale, only versioned. Thus, this call will fail (with a 400 Bad Request error code) if a document already exists in the given record with the given external ID.

POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace, PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace/external/{APP_ID}/{EXTERNAL_ID}  
Replace one document with a new document content. The existing document remains, but is marked suppressed and replaced by the new document.

Medical data cannot be replaced wholesale, only versioned. Thus, this call will fail (with a 400 Bad Request error code) if a document already exists in the given record with the given external ID.

Removing and Archiving Documents

Generally, documents in Indivo cannot be removed, they can only be versioned. However, mistakes happen, and Indivo must deal with these somehow. Also, information eventually is out of date or no longer relevant.

All such changes are encoded in the Indivo API as changes to document status.

POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/set-status  
Change the status of a document. The passed status defines what happens to the specified document:

- **void**: If a document is entered in error, it can be marked as voided to indicate that the data is invalid.
  Only active documents can be voided. Voided documents are still reachable, but their metadata indicates their status, and by default they are not listed in typical document listings.

- **archived**: If a document is no longer relevant, it can be archived so that it doesn’t show up by default. Archival is different from voiding in that an archived document is still considered medically correct, just not particularly relevant anymore.
  Archived documents are still reachable, but their metadata indicates their archival status, and by default they are not listed in typical document listings.

- **active**: An active document is readily usable and will appear in search listings by default. Setting a document to active status will unvoid a voided document, or unarchive an archived document.
GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/status-history  A document can be voided, unvoided, archived, unarchived any number of times. The status change applies to the entire version lineage of a document. The history of statuses, in reverse chronological order, can be obtained using this call.

Relating Documents

It is often useful to relate documents, e.g. annotating a document, re-filling a prescription, connecting diagnoses to an encounter, etc. In Indivo X, these relations can be declared no matter the data type of the underlying document. An image of an X-ray might be related to an XML document that interprets it, but of course there is no room in the image file for a pointer. So all references are stored externally to the documents.

Relationship types are taken from a fixed list, including:

- interpretation
- annotation
- followup

Eventually, full URLs will be supported for relationship types. The fixed list of types will then correspond to http://indivo.org/vocab/documentrels#{rel_type}.

In the following calls, {DOCUMENT_ID} is the document being interpreted, and {OTHER_DOCUMENT_ID} or the POST content is the interpretation.

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/{OTHER_DOCUMENT_ID}  Create a new relationship of type REL_TYPE between the two passed documents.

POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/, PUT /records/{RECORD_ID}  Create a new document and immediately relate it to an existing document, possibly assigning an external id to the newly created document.

Medical data cannot be replaced wholesale, only versioned. Thus, this call will fail (with a 400 Bad Request error code) if a document already exists in the given record with the given external ID.

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/  List all documents related to the passed document by the relationship REL_TYPE.

DOCUMENT_ID is the interpreted document, and the calls return all interpretations (that are of type REL_TYPE) of that document.

Special Documents

Demographics documents are special in that there should only be one per record, and they should be easy to find.

See also:

Indivo Document Demographics Schema  The XML Schema for Indivo Demographics Data

GET /records/{RECORD_ID}/demographics, GET /carenets/{CARENET_ID}/demographics  Fetch demographics from a carenet or record. Depending on the requested response_format return data is formatted as SMART RDF/XML (default), SDMJ, or SDMX.

PUT /records/{RECORD_ID}/demographics  Update or create demographics on a record.

2.1.5 Messaging and Notifications

Indivo supports a lightweight notification framework as well as a heavier message inbox. For more information, see Messaging and Notifications.
Messaging

GET /accounts/{ACCOUNT_ID}/inbox/  List available messages. By default, only non-archived messages are returned.

GET /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}  Fetch a single message.

POST /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}/archive  Archive a message.

POST /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}/accept  Accept a message attachment. A user can accept an attachment from a message into their medical record. This creates a new document on their record containing the contents of the attachment.

POST /accounts/{ACCOUNT_ID}/inbox/  Send a message to an account.

POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}  Send a message to a record. Messages to records can have attached documents (specified by the num_attachments parameter) which then need to be uploaded separately. The message isn’t delivered until all of its attachments are uploaded.

Since Accounts, not Records, are the users who log into the system to view messages, there is no way to view messages in a record’s inbox. Rather, when a message is sent to a record, every account authorized to view the message is sent a copy of the message, which they can retrieve via their account inbox.

POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}  Upload an attachment to a message.

Notifications

Notifications are intended to be a lightweight system for applications to alert users of activity in the application. This is especially relevant for apps that use sharing functionality: an app might want to notify other users of the app about a given user’s activity in it. UI apps should display these notifications in a twitter-feed like interface (our reference UI call it the ‘healthfeed’).

POST /records/{RECORD_ID}/notifications/  Send a notification to a record. As with inbox messages, notifications are propagated to the accounts that are authorized to view the record.

GET /accounts/{ACCOUNT_EMAIL}/notifications/  List available notifications.

2.1.6 Application-Specific Storage

Application-specific storage is meant for bookkeeping by individual applications that is not specific to any given record. These documents can be deleted, since they are not part of any permanent medical record. All application-specific storage API calls behave like the prior document API calls, though the documents are accessible only to the application in question. Most implementations of the Indivo API will likely impose a quota on Applications to ensure they do not store large amounts of data in the application-specific storage. This quota may be application-specific, depending on what the application is approved to do.

Application-specific storage calls, since they don’t correspond to any given record, are all 2-legged oAuth calls.


GET /apps/{APP_ID}/documents/{DOCUMENT_ID}  Fetch a single application-specific document.

GET /apps/{APP_ID}/documents/{DOCUMENT_ID}/meta, GET /apps/{APP_ID}/documents/external/{EXTERNAL_ID}/meta  Fetch metadata about a single application-specific document, by its internal or external id.

POST /apps/{APP_ID}/documents/, PUT /apps/{APP_ID}/documents/external/{EXTERNAL_ID}  Create an application-specific document, possibly assigning it an external id.
As this is application-level storage, making this call with an external id will overwrite any existing document with the same external id.

**PUT /apps/{APP_ID}/documents/{DOCUMENT_ID}/label**  
Update the label of an application-specific document.

**DELETE /apps/{APP_ID}/documents/{DOCUMENT_ID}**  
Delete an application-specific document. Since these documents do not contain medical data, deleting them is acceptable.

### 2.1.7 Record-Application-Specific Storage

Record-application-specific storage is meant for bookkeeping by individual applications. These documents can be deleted, since they are not part of the permanent medical record. All record-application-specific storage API calls behave like the prior document API calls, though the documents are accessible only to the application in question. Most implementations of the Indivo API will likely impose a quota on Applications to ensure they do not store large amounts of data in the record-application-specific storage. This quota may be application-specific, depending on what the application is approved to do.

Record-application-specific storage calls are all 3-legged OAuth calls.

**GET /records/{RECORD_ID}/apps/{APP_ID}/documents/**  
List record-application-specific documents. Supports order by document metadata fields (see *Indivo Document Metadata Schema*).

**GET /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID}**  
Fetch a single record-application-specific document.

**GET /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID}/meta, GET /records/{RECORD_ID}/apps/{APP_ID}/documents/external/{EXTERNAL_ID}/meta**  
Fetch metadata about a single record-application-specific document, by its internal or external id.

**POST /records/{RECORD_ID}/apps/{APP_ID}/documents/**, **PUT /records/{RECORD_ID}/apps/{APP_ID}/documents/external/**  
Create a record-application-specific document, possibly assigning it an external id.

As this is record-application-level storage, making this call with an external id will overwrite any existing document with the same external id.

**PUT /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID}/label**  
Update the label of a record-application-specific document.

**DELETE /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID}**  
Delete a record-application-specific document. Since these documents do not contain medical data, deleting them is acceptable.

### 2.1.8 Processed Medical Reports

Indivo processes documents into medical reports. Each report can be altered by the basic paging mechanism or the more complex query interface described above. Over time, new reports may be introduced. For now, we define these as the minimal set of reports. Fields supported by individual reports for the querying interface may be found here. Response formats correspond to the *Indivo Reporting Schema*, and individual reports fit their individual datatype’s schema (see *Medical Documents*). If a report is accessed via a carenet, only documents that are shared into the carenet will appear in the results.

**GET /records/{RECORD_ID}/reports/minimal/equipment/**, **GET /carenets/{CARENET_ID}/reports/minimal/equipment/**  
List equipment for a given record.

**GET /records/{RECORD_ID}/reports/minimal/labs/**, **GET /carenets/{CARENET_ID}/reports/minimal/labs/**  
List lab results for a given record.

**GET /records/{RECORD_ID}/reports/minimal/measurements/**, **GET /carenets/{CARENET_ID}/reports/minimal/measurements/**  
List measurements for a given record.
GET /records/{RECORD_ID}/reports/minimal/procedures/, GET /carenets/{CARENET_ID}/reports/minimal/procedures/  List procedures for a given record.

GET /records/{RECORD_ID}/reports/minimal/simple-clinical-notes/, GET /carenets/{CARENET_ID}/reports/minimal/simple-clinical-notes/  List clinical notes for a given record.

### 2.1.9 SMART API Calls

As Indivo now supports the SMART API, the following calls are now available:

**GET /records/{RECORD_ID}/{MODEL_NAME}/**  Get a SMART RDF list of a patient’s medical data of type `MODEL_NAME`. Available data models are:

- allergies
- encounters
- fulfillments
- immunizations
- lab_results
- medications
- problems
- vital_signs

**GET /apps/{APP_ID}/manifest, GET /apps/manifests/**  Get SMART-style JSON manifests for one or all apps registered with this instance of Indivo.

**GET /ontology**  Get the ontology used by a SMART container

**GET /capabilities/**  Get the SMART capabilities for this instance of Indivo.

**GET /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences**  Get account preferences for a specific application.

**PUT /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences**  Set account preferences for a specific application. Overrides previous preferences.

**DELETE /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences**  Remove all account preferences for a specific application.

### 2.1.10 Generic Reports

Indivo provides the ability to run ‘generic’ reports over all `data models`. These reports support the `API Query Interface`, and provide an out of the box solution for reporting over core and contributed data models, with the possibility for customization.

**GET /records/{RECORD_ID}/reports/{DATA_MODEL}/**, GET /carenets/{CARENET_ID}/reports/{DATA_MODEL}/  List a patient’s medical data.

### 2.1.11 Coding Systems

A number of Indivo documents contain coded values. These can be based on UMLS, SNOMED, etc. Indivo provides a generic API for looking up coded values. This API is particularly built to support live autocomplete in JavaScript.

**GET /codes/systems/**  List available coding systems. Return data is in JSON format.
GET /codes/systems/{SHORT_NAME}/query  Search a coding system for a value.

2.1.12 Autonomous Apps API

Autonomous user applications are unlike standard user apps in that they may not have a user interface, and require access to records without an active user session. In order to authenticate against Indivo on behalf of records at any time, autonomous apps may make the following calls:

GET /apps/{APP_ID}/records/  Return a list of records which have enabled the app, and to which (therefore) the app can authenticate and acquire access.

POST /apps/{APP_ID}/records/{RECORD_ID}/access_token  Retrieve a valid access token providing the app with access to a record. This call will only succeed if the app is autonomous, and if the record has enabled the app.

Using this call, an autonomous app can retrieve a valid access token for any record on which it is enabled, without an active user session.

2.1.13 Administrative API

Admin applications have access to Indivo’s administrative API, which enables control and setup of records and accounts.

Account Administration

GET /accounts/{ACCOUNT_ID}  Get information about an account. The account_id must be in the form of an email address.

POST /accounts/{ACCOUNT_ID}/info-set  Set information about an account, including the name and contact email of the account holder.

GET /accounts/search  Search for accounts by name or contact email.

GET /accounts/{ACCOUNT_EMAIL}/records/  List available records on an account. Supports order by label.

POST /accounts/  Create an account.

The primary and secondary secret arguments are optional and are used for helping the user initialize their account securely. A primary secret is sent directly by Indivo X server to the user at their ACCOUNT_ID email address in the form of a URL with an embedded secret. A secondary secret is generated by Indivo X and made available to the admin application using the GET /accounts/{ACCOUNT_ID}/secret call for the account. If it is asked for in this call, it is required at account activation time right after the user clicks on the activation URL (aka the primary secret). A secondary secret makes sense only if a primary secret is also requested. That’s why it’s called “secondary.”

POST /accounts/{ACCOUNT_ID}/authsystems/  Add an authentication system to an account.

Accounts initially have no “authentication systems” attached to them. Over time, Indivo accounts will be usable with OpenID and other authentication systems. An account needs to enabled for each authentication system that we want to use for that account. The default system is “password”. Thus, this call, when used with the “password” system, will set up the password and username for a new user.

POST /accounts/{ACCOUNT_ID}/secret-resend  Resend an account’s initialization URL (which contains the primary secret for the account). This is useful if the account holder loses the original email.
POST /accounts/{ACCOUNT_ID}/forgot-password  Reset an account when its password is forgotten.

If a password is forgotten, the solution is to reset the account and email the user as with their initialization email. This will prevent logins until the new initialization URL is clicked, and the new password is entered.

This could be accomplished with separate calls to POST /accounts/{ACCOUNT_ID}/reset, which sets the account state to uninitialized and resets the account secrets, and POST /accounts/{ACCOUNT_ID}/secret-resend, but this call combines both actions.

Note that this call resets both the primary and secondary secrets. The user will need to be given this secondary secret in a channel other than email. If a User Interface Application performed this reset, then the secondary secret should display on screen while the primary secret is automatically sent by email. The user interface could obtain the secondary secret (which is short) by calling GET /accounts/{ACCOUNT_ID}/secret, but the call to POST /accounts/{ACCOUNT_ID}/forgot-password returns the secondary secret to avoid the extra call.

POST /accounts/{ACCOUNT_ID}/initialize/{PRIMARY_SECRET}  Initialize a new account.

Initializing an account that has been reset requires both the primary and secondary secrets. The primary secret is sent in the URL, and the secondary secret should be collected by the user interface. Specifically, the recommended process is:

- Indivo Backend server sends the reinitialization URL to the user as:
  
  INDIVO_UI_APP_LOCATION/account/initialize/account_id/primary_secret

- An Indivo UI App checks that the requested account is indeed in uninitialized state and prompts the user for his secondary secret (which the user knows simply as the "secret") and his desired username and password.

- The Indivo UI App initializes the account using this call.

- The Indivo UI app sets up the account with the built-in password authsystem using the username/password provided by the user and the API call POST /accounts/{ACCOUNT_ID}/authsystems/.

POST /accounts/{ACCOUNT_ID}/set-state  Set an account’s state. Possible account states are:

- uninitialized: an account that has been created by an administrative application and has not been activated by the user yet (with their confirmation URL and code).
- active: a normal active account.
- disabled: an account locked because of too many failed login attempts.
- retired: an account that is no longer in use.

POST /accounts/{ACCOUNT_ID}/authsystems/password/set  Force an account’s password to a new value. This should be used only in the context of an account reinitialization.

POST /accounts/{ACCOUNT_ID}/authsystems/password/change  Allow a user to change an account password. The given old password must be correct for this change to succeed. This is a 3-legged call, since the user is the one driving this interaction (unlike POST /accounts/{ACCOUNT_ID}/authsystems/password/set, wherein the admin app is forcefully setting a password).

GET /accounts/{ACCOUNT_ID}/primary-secret  Fetch an account’s primary secret. This should be used very sparingly as the primary secret should rarely be seen outside of the Indivo backend.

Record Administration

GET /records/{RECORD_ID}  Get info about a single record.

GET /records/search  Search Indivo for existing records by record label.
GET /records/{RECORD_ID}/owner  Get the owner of a record

GET /records/{RECORD_ID}/apps/  List applications attached to a record. Supports order by name.

POST /records/{RECORD_ID}/apps/{APP_ID}/setup  Prime a record with a user app. This sets up an app to run against a record without user consent. It should be used only in cases where obtaining consent is impossible or unnecessary (i.e., at a hospital installation of Indivo, this call could be used to prime all new records with the syncer application that pulls data into Indivo from the hospital EMR).

PUT /records/{RECORD_ID}/owner  Set the owner of a record.

PUT /records/{RECORD_ID}/apps/{APP_ID}  Enable an app to run against a record. This gives the app access to the entire record.

DELETE /records/{RECORD_ID}/apps/{APP_ID}  Remove a user app from a record.

2.1.14 Indivo Chrome / User Interface API

These API calls are reserved for the UI server, which is deeply trusted to authorized other applications, proxy the user’s credentials, etc. It’s only a separate server for modularity, otherwise it has the same level of trust as the backend Indivo server.

POST /oauth/internal/session_create  Create a web-session for a user. This call returns a session token that can be used to authenticate 3-legged calls on behalf of the user for the duration of a standard web session (30 minutes by default)

POST /oauth/internal/request_tokens/{REQUEST_TOKEN}/claim  Claim a request token on behalf of a user. Before a request token can be viewed at all, it has to be claimed by a user. This ensures that a request token can’t be partially used by one user and completed by another.

The session-based chrome authentication will indicate to the backend which Account to associate with this request token. Once this call has been made for a request token, a second call with different session authentication will fail. (A second call with the same user authentication will be just fine, we don’t want a reload to cause a problem.)

If the request token is bound to an Indivo record (because the PHA knew it was authorizing for a given record), and the claimant does not have administrative rights over the record, this call will fail and the request token will be invalidated.

GET /oauth/internal/request_tokens/{REQUEST_TOKEN}/info  Retrieve information about an OAuth request token.

When authorizing a request token, the Indivo UI needs to know what that token represents. Once the token is claimed, the request token yields information via this call.

This call can only be called with session authentication matching the Account which claimed the request token earlier.

If a record_id is present in the response data, then the kind element is also present and indicates:

- new: a new request for a PHA that has not been authorized for this record yet
- same: a request for a PHA that is already attached to the record and no new permissions are requested
- upgrade: a request for a PHA that is already attached to the record but that is asking for more permissions or more permissive usage of the data.
In the same case, the Chrome UI is allowed to immediately approve the request token. In other cases, the Chrome UI must explain to the user that new permissions or rights are being granted and prompt the user for approval.

**POST /oauth/internal/request_tokens/{REQUEST_TOKEN}/approve** Approve a request token on behalf of a user.

If a user approves an app addition, then the Chrome UI server needs to let the backend know. This call, if it succeeds with a **200 OK**, will return the location to which the user’s browser should be redirected:

```
location={url_to_redirect_to}
```

This call’s session authentication must match that which claimed the request token. The `record_id` is the record to which the user is attaching the application (i.e. my child’s record, not my own.) If the request token was pre-bound to a record, this `record_id` parameter must match, or this will throw an error.

**POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials** Get credentials for Connect-style Authentication for a user app, and authorize them on behalf of an account.

This call will return tokens that can be used to sign future API calls by the user app, proxied by the UI.

**GET /accounts/{ACCOUNT_ID}/check-secrets/{PRIMARY_SECRET}** Check the primary and secondary secrets of an account.

**GET /oauth/internal/surl-verify** Verify a signed URL.

In some cases, an Indivo app will sign a URL that directs the user to the Indivo UI. A prime example is the use of Indivo Chrome widgets, i.e. the Document Sharing widget, that apps can embed within their user interface to reuse functionality from Indivo Chrome. A signed URL looks like this:

```
/widgets/WidgetName?param1=foo&param2=bar&surl_timestamp={TIMESTAMP}&surl_token={TOKEN}&surl_sig={SIGNATURE}
```

The signature contained in `surl_sig` is effectively a signature on the rest of the URL. The signature algorithm is as follows:

1. An app, with oAuth access token `TOKEN` and oAuth access token secret `SECRET`, wishes to sign a URL.
2. The app generates the SURL secret that corresponds to this access token as follows:

   ```
   <SURL_SECRET> = HMAC(<TOKEN_SECRET>, "SURL-SECRET")
   ```

   using base64 encoding, where the idea is to actually sign the string “SURL-SECRET” to obtain the SURL secret itself.
3. this SURL secret is then used to sign the URL, first by appending a timestamp, the SURL token, and then computing the signature:

   ```
   <SURL_SIG> = HMAC(<SURL_SECRET>, "/widgets/WidgetName?...&surl_timestamp=<TIMESTAMP>&surl_token=<TOKEN>
```

in base 64, then appending it as a query parameter `surl_sig`.

### 2.1.15 Sharing

**Overview**

We want to simplify sharing. Indivo has two main mechanisms for sharing patient records with other accounts: **Full Shares** and **Carenets**.

**Full Share** A user may choose to share the entirety of their record with another account. The recipient account will then have access to all data (past, present, and future) contained in the record, and will be able to run any apps
that have been bound to the record. The recipient of a full share will also be able to add new applications to the
record and run them against data in the record.

Similarly, a user may choose to add an application to their full record. This effectively creates a ‘full share’ of
the record with that application: the app has access to all data in the record.

As an example, a teen user of Indivo might choose to set up a full share of his / her record with a parent of
guardian.

**Carenets** Full shares are not very flexible: they are an all or nothing proposition. In cases where sharing data ap-
propriately requires more granularity or complexity, Indivo provides **carenets**, which allow a record to specify
groups of **accounts** and **apps** that all have transparent access to whatever **data** the record shares into the carenet.

By default, each record will have 3 simple carenets: physicians, family, and work/school.

As an example, a patient might create an ‘exercise’ carenet, into which they place:

- **data**: blood-pressure readings, pedometer output, and other data associated with maintaining a healthy
  lifestyle.

- **apps**: blood-pressure viewers, exercise-trackers, and other apps that help the patient organize and interact
  with their exercise data.

- **accounts**: The patient’s Primary Care Physician, personal trainer, friends, or any other person with an
  interest in helping the patient develop healthy exercise habits.

Now anyone on the **accounts** list can log into Indivo and run any app on the **apps** list against any data on the
**data** list.

Data can be placed into carenets individually, or autoshares by Document type. Users can override the type-
auto-sharing on a document-by-document basis.

**Autoshare** Documents of a certain type can be auto-shared, so that they are added to a carenet automatically when
they are added to the record. When auto-share preferences are set for a type of document within a given carenet,
these preferences apply to all documents that do not have an explicit sharing preference declared on them.

**Nevershare** A user should be able to ask that a specific document be “never shared”. This flag prevents any sharing,
no matter what the auto-share rules may be.

**Authorization into a CareNet**

When an app is added, it is normally given, along with its oAuth token, an **xoauth_indivo_record_id** that
the token corresponds to. If the app is added to a carenet instead of a record, the app will receive instead an
**xoauth_indivo_carenet_id**.

**Carenet-aware API calls**

Many of the document and reporting calls that can be made on `/records/{RECORD_ID}` can be made on
`/carenets/{CARENET_ID}`. Where applicable, such calls have been listed throughout this document.

Importantly, carenets are (at present) **READ-ONLY**. Accounts placed in carenets may view any data in the carenets,
but we have not implemented any calls for them to modify or add to that data. In the future, carenets will be write-
capable.
Sharing API

Full Shares

GET /records/{RECORD_ID}/shares/ List accounts with which a record has created full shares. This call also lists user apps that have access to the full record, as such apps have the same access to data as an account with a full share.

POST /records/{RECORD_ID}/shares/ Create a new full-record share with an account. The role_label is currently nothing more than that: a label. The label will come back in a call to GET /records/{RECORD_ID}/shares/.

DELETE /records/{RECORD_ID}/shares/{ACCOUNT_ID} Delete a full share.

Basic Carenet Calls

GET /records/{RECORD_ID}/carenets/ List existing carenets on a record.

GET /carenets/{CARENET_ID}/record Fetch basic information about the record that a carenet belongs to.

POST /records/{RECORD_ID}/carenets/ Create a new carenet on a record.

POST /carenets/{CARENET_ID}/rename Rename a carenet.

DELETE /carenets/{CARENET_ID} Delete a carenet. This will unshare all of the data in the carenet with all users and apps in the carenet.

Data in Carenets

Carenets are useless until data has been shared into them. Data can be shared explicitly at the granularity of individual documents, or implicitly at the granularity of document type.

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID} Place a document in a carenet.

When a document is explicitly shared with a carenet, it is no longer tied to the auto-sharing rules for that carenet. However, auto-sharing rules with other carenets still apply.

DELETE /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID} Remove a document from a carenet

When a document is explicitly Unshared from a carenet, it is no longer tied to the auto-sharing rules for that carenet. However, auto-sharing rules with other carenets still apply.

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/ List carenets where a document is present

The mode attribute indicates how this document is shared. explicit means that the sharing preferences for this document are explicitly set. bytype indicates that it was auto-shared by document type. Other modes may be enabled in the future.

The value attribute indicates a negative share with a carenet, meaning that the user explicitly wants this document not shared with this carenet, even if auto-share rules would otherwise share it. Obviously this only makes sense for explicit carenet-shares.

POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}/autoshare-revert Revert a document to auto-share rules. This means that, for this carenet, this document reverts to automatic sharing rules. This might mean a removal of the share with this carenet, or an addition, or no effect. However, from this point on, the record-wide rules apply.
Indivo X Documentation, Release 2.0.0

**Warning:** This call has not yet been implemented.

**GET** /records/{RECORD_ID}/autoshare/bytype/ List auto-sharing preferences for a given document type with a record. This call returns a list of carenets into which the document type is auto-shared.

**GET** /records/{RECORD_ID}/autoshare/bytype/all List all auto-sharing preferences for a record. This call returns a list of document types with the carenets into which each type is auto-shared.

**POST** /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/set Add an auto-share of a given document type into a given carenet. This share applies to all documents that do not have an explicit sharing preference declared on them.

**POST** /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/unset Remove an auto-share for a given document type from a given carenet.

**PUT** /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare Set the never share flag on a document. A user should be able to ask that a document be “never shared”. This flag prevents any sharing, no matter what the auto-share rules may be.

**DELETE** /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare Remove the never-share flag on a document.

**Apps in Carenets**

Users needs to be able to place apps inside carenets in addition to documents, so that other accounts can run the applications. There are no issues with read/write permissions here, as permissions are associated with the accounts in a carenet, not the apps.

**GET** /carenets/{CARENET_ID}/apps/ List all apps in a carenet.

**PUT** /carenets/{CARENET_ID}/apps/{PHA_EMAIL} Add an app to a carenet.

**DELETE** /carenets/{CARENET_ID}/apps/{PHA_EMAIL} Remove an app from a carenet.

**GET** /carenets/{CARENET_ID}/apps/{APP_ID}/permissions Check an app’s permissions within a carenet. Since permissions are currently handled on accounts, not apps, this call will always indicate that the app has full permissions on the carenet.

**Warning:** This call has not yet been implemented.

**Accounts in Carenets**

Users needs to be able to place other accounts inside carenets so that they can share data and apps. When accounts are added to a carenet, they are assigned read/write permissions, which define what actions they can take on data in the carenet.

**GET** /carenets/{CARENET_ID}/accounts/ List all accounts in a carenet.

**POST** /carenets/{CARENET_ID}/accounts/ Add an account to a carenet.

**DELETE** /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID} Remove an Account from a carenet.

**GET** /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID}/permissions Check an account’s permissions within a given carenet. This call will return a list of document types, and whether the account may write to each one within the given carenet.
or now, the document type is always ‘*’, since read/write permissioning is currently more granular than at the carenet level. We may eventually permit permissioning by document types within a carenet, in which case this all will be more informative.

Building Carenet-aware Apps

Indivo apps are given a record_id and an access token that matches that record to read and write documents, read reports, annotate, etc. In a sharing scenario, apps must become carenet-aware.

Requirements

- An app should be easily placed within any number of carenets, i.e. physicians and family, but not work/school.
- When an app is activated on a given record, it must have access to no more data than the user who activated it. For example, if the owner selects the app, then the app may have access to the entire record. If the owner’s school nurse activates the app, the nurse should have access to only the data that is in the work/school carenet.
- There may be a need to further constrain an app, so that even if the owner activates the app, it should not be able to see every data type, or may be constrained to one of the carenets anyways. This is DEFERRED for now.
- We must not depend on app developers to properly partition information. If an app is active in both the Family and Physicians carenets, and knows that the record_id is the same in both cases, it may well intermix data without realizing it. This would be bad. We need to make it harder for apps to hurt the user.

Scenario

Alice owns her Indivo record and has shared it with Bob, her HR representative at work, placing Bob in the “Work/School” carenet. Alice is pregnant but does not wish to reveal this information to her co-workers just yet. She has added the “Pregnancy Tracker” app to her record, making it visible to her Family and Physician carenets, but not to to her Work/School carenet. Alice has a history of depression, information which she has shared with her Physicians, but not with her Family.

Visible Apps

The “Pregnancy Tracker” app has been added to the Family and Physicians carenets, but not the Work/School carenet, so Bob cannot even see the application when he visits Alice’s record. This is enforced by the Indivo platform itself.

Activating and using an App

Charlie, Alice’s father, is eager to check up on his future grandchild’s progress. He logs into Indivo, selects Alice’s record. He sees “Pregnancy Tracker” because that app is visible to the Family carenet. He launches the app, and uses its functionality to track Alice’s progress, her fetus’s growth, her blood tests, etc. The process when launching the app is:

- Clicking on the app directs the IFRAME to the start_url for the pregnancy tracker. The app must receive an indication of which record is being accessed at this point. This cannot be the record_id alone, and we may not even want to include the record_id at all, otherwise the app might confuse this data with that accessible to Physicians later on. Thus, instead of passing record_id to the IFRAME, Indivo passes only carenet_id.
- The OAuth process begins with the carenet_id only as part of the request for a request token.
- Indivo checks that the logged-in-user has the right to access this carenet, and if so authorizes the token.
- The token is bound to that carenet only, and cannot be used on any other carenet.
- The app can make requests to

/\carenets/{carenet_id}/documents/...
without using the record_id at all. It doesn’t need to know the record_id.

• When the app is later activated by a Physician, who does have access to Alice’s history of depression, the app gets a different carenet_id, and from that carenet has access to the documents including mental health.
• This is not fool-proof: we still probably need to give the app access to some record information that will yield a unique identifier using the name, DoB, etc... but at least the default behavior for the app will not allow error-prone tracking across carenets.

OAuth Mechanics

We start with:

• A CarenetAccount row that shares a record’s carenet with another account
• A Share row that indicates that an app has access to the record
• A CarenetPHA row that makes the app available in the carenet.

The OAuth process is then:

• PHA requests a request token with a carenet_id instead of a record_id as parameter.
• PHA needs to have a share into the record or into the specific carenet for this to succeed.
• The request token needs to keep track of the carenet, because the Share might be for the whole record.
• The user approving the request token should be in the carenet in question.
• The access token already stores the account of the person it’s proxying for, so that should be enough.

2.1.16 Auditing

As Indivo will be installed at HIPAA-compliant hospital sites, it is important that it be able to track system usage. The auditing system logs all incoming requests to Indivo that use the Indivo API. To learn more about auditing in Indivo, see the audit system’s documentation.

GET /records/{RECORD_ID}/audits/query/ Query the audit system. This call allows for general queries over the audit logs. The query is specified in the parameters via the API Query Interface, and returns results in the style of Medical Reports.

All subsequent calls are deprecated, but maintained (for now) for backwards compatibility.

GET /records/{RECORD_ID}/audits/ List all audit log entries where a given record was accessed.

  Deprecated since version 1.0.0.

GET /records/{RECORD_ID}/audits/documents/{DOCUMENT_ID}/ List all audit log entries where a given document was accessed.

  Deprecated since version 1.0.0.

GET /records/{RECORD_ID}/audits/documents/{DOCUMENT_ID}/functions/{FUNCTION_NAME}/ List all audit log entries where a given document was accessed via a given internal Django view function.

  Deprecated since version 1.0.0.
2.2 Architecture Overview

This document provides a basic overview of the Indivo X system architecture. This document should be read before continuing on to Indivo Authentication and the Indivo API.

2.2.1 Basic Indivo Concepts

Indivo Record: the complete set of medical information stored by Indivo about a single individual.

Indivo Account: a username/password to log into Indivo. One account may be able to access any number of Indivo Records, and one Indivo Record may be accessible by multiple Indivo Accounts.

Indivo Document: a piece of medical information stored in an Indivo Record.

2.2.2 Components

Indivo X comprises multiple components, each running as its own web server. Small installations may choose to install multiple components on a single physical server. The Indivo X Server is the core of the system; other components, including the Indivo User Interface, can be easily substituted by custom implementations.

---

Indivo X Server

For a given Indivo installation, the Indivo X server:

- stores all Indivo account information, as well as the medical records and documents,
• is responsible for authentication and authorization before granting access to Indivo data,
• exposes an API for access by administrative and user applications, and by the Indivo User Interface.

**Indivo User Interface / Indivo Chrome**

The Indivo User Interface, also known as the “Indivo Chrome”, implements the web-based visual interface that an Indivo user will view and use. The branding/colors/details of the user interface are all controlled by the Indivo Chrome. Indivo Chrome connects to Indivo X using the standard Indivo API, including some specific calls accessible only to the Chrome component.

Indivo X will ship with a default implementation of the Indivo Chrome which can be customized while maintaining a clean interface to the Indivo X API. Customizations are encouraged for re-branding or for entirely different devices, e.g. iPhone.

(The term “Chrome” is often used to describe the visual portions of a web browser that are part of the web browser itself, and not part of the web site content, e.g. the back button. Here, with “Indivo Chrome”, we mean the Indivo user interface that is part of the core Indivo service, not part of a user application that extends Indivo functionality.)

**Administrative Application**

An Admin Application can connect to Indivo X and

• create new Indivo accounts and records
• reset of passwords
• manage ownership of records, i.e. assigning an account as the owner of a record.

An admin application cannot access medical data, it can only manage a record’s metadata. An admin app is thus ideal for a hospital administrator, an Indivo help-desk staffer, a research administrator, etc.

**User Application / Personal Health Application**

A user application, or Personal Health Application, is an application that Indivo users manually add to their record to provide incremental functionality. Examples of PHA functionality include:

• Diabetes management
• Genomic data display
• Clinical trial matching and messaging

User applications generally provide a web interface to the Indivo user, while connecting to the user’s Indivo record directly with the Indivo X Server. Users are fully in control of what data a user application can access. They can, at any time, change those permissions or remove the application entirely. Thus, an Indivo user application connects to Indivo in much the same way that a Facebook application connects to Facebook.

**2.2.3 Communication Protocols**

All communication between components is over HTTPS, with an API that abides by the REST design philosophy. Authentication is via oAuth.

More about Indivo Authentication.

More about the Indivo API.
2.3 Indivo Authentication

As of version 2.0, Indivo now permits three methods of authentication:

- *Traditional oAuth*
- *In-browser Connect Authentication* (a la. SMART CONNECT)
- *Pre-generated REST Authentication* (a la. SMART REST)

2.3.1 Indivo oAuth

Here, we provide details of exactly how Indivo uses oAuth, in particular the options that are supported and those that aren’t.

**Specifics of our oAuth implementation**

OAuth Section 5.2 defines three possible approaches to sending OAuth Protocol Parameters. In Indivo, we use exclusively the HTTP authorization header, as defined by the OAuth specification, as the preferred method. We allow `oauth_callback` and `oauth_verifier` to be provided as POST parameters, only because some client libraries that are not oAuth 1.0a-compatible cannot otherwise connect.

We require all changes suggested by the oAuth 1.0a revision:

- When asking for a request_token, the PHA *must* provide `oauth_callback` as an extra OAuth parameter. This can be a URL where Indivo X will send the user after token authorization, or it can be `oob` to use the default URL specified by the PHA at registration time.
- When responding to a request token, Indivo X will include `oauth_callback_confirmed=true`
- A PHA’s callback URL should accept both `oauth_token` and `oauth_verifier` parameters.
- When exchanging a request token for an access token, the PHA must provide the corresponding `oauth_verifier` parameter in addition to all existing parameters.

In addition, we implement the following constraints:

- As per the OAuth recommended approach, all OAuth token setup and exchange calls use the POST method. The Indivo server will respond with a 405 Method Not Allowed error code to any GET request against its OAuth protocol URLs.
- The `oauth_version` parameter is mandatory. Every PHA request should include the `oauth_version` parameter. The only supported value in Indivo at this point is `oauth_version=1.0`.
- Given an Indivo Server running at https://INDIVO_SERVER, the OAuth URLs are defined as follows:
  - Request Token URL: https://INDIVO_SERVER/oauth/request_token
  - User Authorization URL: Since there is a UI component to enabling User Authorization (i.e., we have to obtain their consent), Indivo_Server does not explicitly offer a User Authorization URL via its API. Individual UI apps can (with a valid user session) authorize tokens on behalf of users with a call to POST /oauth/internal/request_tokens/{TOKEN}/approve, but each individual UI-app implementation will provide a different app-facing User Authorization URL. We recommend using https://UI_SERVER/oauth/authorize?oauth_token={token}, which is the URL used by our reference UI-app implementation.
  - Access Token URL: https://INDIVO_SERVER/oauth/access_token
• oAuth defines a number of default signature methods and leaves open the possibility of using other signature methods. Indivo supports only one request signature method: HMAC-SHA1. Support for RSA-SHA1 may eventually be added.

Body and Content-Type

By default, oAuth only signs the body of HTTP requests that are form-url-encoded. Indivo uses the oAuth Body Hash Extension to ensure that raw POSTs, e.g. to send XML documents, become part of the signature. When the body-hash extension is activated, Indivo also expects an additional parameter, oauth_content_type, to certify the content type HTTP header (and prevent content-sniffing attacks.)

User Applications (PHAs)

PHA Registration

A PHA registers with Indivo using an Indivo-installation specific process, at the conclusion of which both Indivo and the PHA agree on:

• The name of the PHA, e.g. “Medical Surveys”
• An oAuth consumer_key and consumer_secret.
• A start_url_template for the PHA, e.g. http://acme.com/indivoapp?record_id={indivo_record_id}&
• A callback_url for the PHA, e.g. http://acme.com/success_after_indivo, which should expect to receive query parameters oauth_token and oauth_verifier.
• Whether the PHA has a web user interface (certain applications that synchronize data have no UI), and whether that PHA is frameable inside Indivo.
• Whether the PHA is autonomous or not, and if it is, why it wants that kind of access to the user’s personal health record.

IMPORTANTLY, the callback_url is the only URL that Indivo will return the user to after a successful PHA attachment. Indivo does not support a custom oAuth callback URL.

Autonomous Apps

An autonomous app is one that wants to access the user’s record while the user is not connected. PHAs that qualify include hospital data connectors, drug-interaction checkers, etc. There are very good reasons for PHAs to access a record while the user is not online, but we want to ensure that users understand the implications, and thus the Indivo authorization pathway looks different depending on whether an app is autonomous or not.

An app must choose to be autonomous at registration time. It must be autonomous for all users, or for none.

An autonomous app accesses the entire record by default, and the user must consent to this. This design choice is meant to prevent medical mistakes for automated apps that, for example, check for drug-drug interactions but may fail to notify the user if they have only partial data access. An autonomous app thus triggers the appropriate authorization screen that warns the user about the long-term, autonomous access, displays the app’s reason for requesting this type of access, and simply gives the user a yes/no choice.

Autonomous apps can, in some circumstances, have no user-interface. This might happen if, for example, a hospital connector application sits behind the hospital firewall and connects autonomously to the Indivo record to upload hospital data into the PCHR, but never lets the user connect directly to the app itself. There are two ways, currently, to authorize such an application. The first is via admin-based PHA setup, where an administrative app primes the Indivo record with this app. Alternatively, the UI may allow users to permission autonomous apps without an interface. In

2.3. Indivo Authentication
this case, there is no oauth dance: the user ‘enables’ the app, and the app is then able to acquire access tokens for enabled users directly. In any case, these apps must declare their lack of UI at registration time, much like they declare their being autonomous or not. Only autonomous apps can choose to forgo a UI.

A non-autonomous app, on the other hand, is one that is meant to be used by whoever is logged in and has access to the record in question. Depending on which user has launched the app, the app’s permissions might differ. For example, when Alice uses the Problems App within her record, she should see “all” of her problems. However, when Bob, her co-worker, uses the Problems App to view Alice’s record, he should see only those problems which Alice has chosen to let him see. Thus, a non-autonomous app exists purely to proxy a human user’s clicks and perform some visualization / data entry assistance functionality. Non-autonomous apps are thus constrained to a carenet at the time that the user clicks on the app name to launch it. When Bob launches the Problems App on Alice’s record, the Problems App receives an access token that is constrained to Alice’s “Work” carenet, and the app can only access the problems Alice has made available within her Work carenet. All access tokens for non-autonomous apps are valid only for the duration of a web session.

Connecting a PHA to a Record

A user opts to add a PHA to her Indivo record by enabling it in the UI. At any subsequent point, when the user attempts to use the PHA (i.e. by clicking on it in the UI), she is sent to the PHA’s start URL with the indivo_record_id filled in. The PHA may present informational content if it so desires, then is expected to begin the OAuth authorization process. When the PHA begins the OAuth process, it should do so with the indicated indivo_record_id that it received when its start_url was accessed.

Obtain a Request Token   A PHA begins its access request for a user when the user visits the PHA’s start URL. While the user’s browser awaits a response, the PHA obtains from the Indivo Server a request token. This is accomplished by issuing a signed POST 2-legged OAuth request to the Request Token URL:

https://INDIVO_SERVER/oauth/request_token

with optional form parameter indivo_record_id. Again, if the PHA was accessed via its start_url with the Indivo record ID filled in, it should use this record ID at this point in obtaining the request token. Otherwise, the user interface will be thoroughly confusing.

This call returns an OAuth token:

oauth_token={token}&oauth_token_secret={secret}

The PHA is expected to store the Request Token and its correspondence to this specific user, likely in the web session.

Authorize the Request Token   Once it has obtained a request token, with the user’s browser still waiting for a response, the PHA responds by redirecting the user’s browser to the User Authorization URL on an Indivo UI app, indicated in the request token response above, or by default:

https://UI_SERVER/oauth/authorize?oauth_token=<REQUEST_TOKEN>

with the request_token as a URL query parameter named oauth_token. Note how this URL is not a signed OAuth request. This step is simply a redirection of the user’s browser to her Indivo account in order to prompt for and obtain authorization.

Indivo prompts the user to authenticate if she isn’t already logged in. Indivo then associates the request token with this user, and only this user can proceed with this specific request token. It is an error for a PHA to attempt to reuse request tokens, and Indivo will prevent this from happening.

Indivo then presents the user with the details of the PHA’s requested permissions.
The user can choose to cancel the process, in which case no further requests are issued, the PHA is not notified, and the request token is discarded.

**Obtain an Access Token**  If the user agrees to connect with the PHA, Indivo redirects the user browser to the PHA’s `callback_url`, as specified by the PHA at registration time. Appended to this `callback_url` are the `oauth_token`, the request token that identifies this authorization dance, and the `oauth_verifier`. The PHA is encouraged to check that the `oauth_token` matches the token stored in its web-session.

The PHA must now exchange the Request Token for an Access Token. This is accomplished using a 3-legged oAuth POST request, with the request token and secret, to:

https://INDIVO_SERVER/oauth/access_token

In response to this request, the PHA obtains an Access Token, including one of two optional parameters:

`oauth_token=<TOKEN>&oauth_token_secret=<SECRET>&xoauth_indivo_record_id=<RECORD_ID>`

or

`oauth_token=<TOKEN>&oauth_token_secret=<SECRET>&xoauth_indivo_carenet_id=<CARENET_ID>`

This token can then be used by the PHA to make 3-legged oAuth calls to Indivo. The Indivo record ID parameter indicates which record this token is bound to, while the carenet indicates which portion of the system the PHA can access.

**Interact and Re-Auth**  At this point, the PHA has an access token, an access secret, an Indivo record ID, and an Indivo privacy group. These credentials allow the PHA to make calls to the Indivo Server to obtain data from the given Indivo record. If the PHA provides a direct web interface to the user, this UI is delivered inside an IFRAME within the Indivo User Interface.

A few days later, when the user returns to his Indivo record, he can click on any of the PHAs he has already authorized. The PHA, however, does not know immediately who this user is. To communicate the user’s identity to the PHA, Indivo simply re-performs the oAuth dance, setting the IFRAME’s URL to the PHA’s starting point with the prescribed Indivo Record ID. When the PHA redirects the IFRAME to the authorization page, Indivo notices that this record has already authorized the app, and simply redirects the IFRAME immediately to the PHA’s `callback_url`. Thus, a complete oAuth process is re-performed, and the PHA re-obtains an access token, access secret, Indivo record ID and privacy group.

The PHA should never assume that the access token and secret stay the same. The long-term identifier that the PHA should key its data against is the Indivo Record ID.

**Admin Applications**

Admin Applications contact the Indivo X server using 2-legged oAuth only, with just a consumer key and consumer secret.

**Chrome Applications**

Most Indivo developers who only wish to write PHAs can safely ignore Chrome applications. Developers who wish to customize the entire Indivo experience need to understand Chrome apps.

The Indivo Chrome (User Interface) contacts the Indivo X server first using 2-legged oAuth to create a user-specific session using the user’s username and password. Indivo X responds with a fresh oAuth token and secret valid for the length of a typical web session. Then all Indivo Chrome calls to the Indivo X server on behalf of a given user are made as 3-legged calls, using the Indivo Chrome’s consumer key and secret, and the specific session token and secret.
2.3.2 In-Browser Connect Authentication

Connect-style authentication enables user applications running framed within the Indivo UI to make API calls solely using javascript, without having to navigate the OAuth dance. Connect-style authentication works as follows:

- When the Indivo UI app opens a user app within its iframe, it acquires a set of OAuth credentials that allows the UI app to make proxied API calls on behalf of the user app, using the API call `POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials`.

- The Indivo UI app additionally opens a channel to the iframe (using something like `jschannel`), so that the app can make client-side requests directly to the UI app.

- When the app wants to make an API call, it makes an unsigned request (in the client-side javascript) through its channel.

- The UI app receives the request, signs it with the Connect credentials, and passes the request through to the backend Indivo Server.

- Indivo Server processes the request, and sends the results back to the UI app, which in turn passes them through the channel to the user app.

Thus, the user app needs to perform no authentication when making API calls—it merely passes the calls through its channel to the UI app (a process which we’ve already implemented in our javascript client) and receives the results. The security of the call is enforced in the channel, and by the tokens used by the UI Server to authenticate the call.

2.3.3 Pre-generated REST Authentication

Pre-generated REST Authentication enables user applications running framed within the Indivo UI to acquire OAuth access tokens that can be used sign 3-legged API calls (as with standard Indivo OAuth) without having to navigate the OAuth dance. The authentication process is as follows:

- When the Indivo UI app opens a user app within its iframe, it acquires and preauthorizes an access token for the app, using the API call `POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials`.

- The UI app then appends a well-formed OAuth header containing the access token to the user app’s start url. The format of the header (and required parameters) are described in the SMART documentation.

- The user app extracts the access token and indivo record ID from the oauth header, and uses it to sign subsequent API calls.

2.4 Indivo Client Libraries

An Indivo client library is any code (in any language) provided as a standard package to app developers which enables them to make API calls against Indivo without worrying about low-level implementation details such as OAuth signing.

We currently support a few simple libraries: this document lists those libraries, and provides advice for generating new libraries.

2.4.1 Supported Client Libraries

Currently, we have released the following client libraries:
Python Client

Our python client is a simple wrapper around the commonly used and supported Python Oauth2 library for making OAuth-signed REST calls.

Our code is available on github, and documentation is available in the Indivo docs.

iOS Framework

The Indivo iOS Framework is an object-oriented wrapper that provides class-based access to core Indivo data-types using the Indivo API.

The code is available on github, and documentation is available in the Indivo docs.

SMART Clients

Since Indivo now supports the SMART API, apps written using SMART client libraries will also run on Indivo. See their documentation <http://dev.smartplatforms.org> for more information

Note: Make sure the the SMART clients you use support SMART v0.4

2.4.2 Building a Client Library

A client library’s responsibilities are simple: it must be able to sign HTTP requests using OAuth, send them to Indivo Server or an Indivo UI App (for OAuth authorization), and present the results of the requests back to the app developer. Most languages have libraries for doing these things already, so building a new Indivo client library is actually quite simple.

In order to facilitate auto-generation of clients, Indivo provides an api.xml file, which describes all of the calls a complete client should support. This file can be found in the indivo server source code, in indivo_server/api.xml.

The api.xml file should be updated whenever the supported API calls are modified. In order to insure that you have the most up-to-date version of the file, you can run (from a valid indivo_server checkout):

```
python manage.py generate_api_spec
```

Which will update the api.xml file to be consistent with the current codebase.

2.5 Indivo Data Models

2.5.1 Introduction

Data Models in Indivo describe the format in which Indivo represents medical information. They are NOT the same as Schemas, which describe formats that Indivo recognizes as valid input data. Rather, data models describe the final processed state of medical data in Indivo: how data are stored, how they are queryable via the Query API, and how they are returned via the Reporting API.

We also introduce one additional term: Medical Facts. A Fact is one datapoint corresponding to a data model: for example, a latex allergy is a Fact that is an instance of the Allergy data model. Internally, Indivo represents facts as Python objects, so you’ll see us referencing medical facts as fact objects as well.
2.5.2 Defining a Data Model

At its most basic level, a data model definition is just a list of fields and their types. For example, our *Problem data model* is defined as (some fields omitted):

- `date_onset`: Date
- `date_resolution`: Date
- `name`: String
- `comments`: String
- `diagnosed_by`: String

This is pretty simple, and we’d like to enable others add new data models to Indivo just as easily. So we currently allow two formats for defining data models:

**Django Model Classes**

Since our data models are directly mapped to database tables using Django’s ORM, they are most effectively represented as Django Models. Django has a flexible, powerful method for expressing fields as python class attributes, so data models defined in this way can harness the full capabilities of the Django ORM. Of course, representing data models in this way requires some knowledge of python. For a full reference of Django models, see Django models and Django model fields.

One important Indivo-specific note: when defining Django Model Classes, make sure to subclass `indivo.models.Fact`, which will ensure that your class can be treated as a data model. For example, your class definition might look like:

```python
from indivo.models import Fact
from django.db import models

class YourModel(Fact):
    your_field1 = models.CharField(max_length=200, null=True)
    ...

    # Additional fields here
```

**Custom Django Model Fields**

For modeling medical data, Indivo provides some custom Field Subclasses. These fields represent their data as multiple separate database fields, with names formed from the original field’s name and some appended suffixes (see the classes below for some examples). You should use these fields as if they were any other Django Model Field:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import YourFavoriteFieldSubclass

class YourModel(Fact):
    normal_field = models.CharField(max_length=200, null=True)
    special_field = YourFavoriteFieldSubclass()
```

Now YourModel has both a standard CharField, and also other fields defined by the Field Subclass. We define the following Field Subclasses:
class `indivo.fields.CodedValueField` *(Type)*
A field for representing coded data elements.

Creating a CodedValueField named ‘value’, for example, will (under the hood) create thee fields:
- `value_identifier`, the system-specific identifier that represents the element (i.e. an RXNorm CUI)
- `value_title`, the human-readable title of the element
- `value_system`, the coding system used to represent the element

When describing instances of your model (either when defining a *transform output* or when referencing fields using the *Indivo Query API*), you must refer to these field names, not the original *value* field name.

class `indivo.fields.ValueAndUnitField` *(Type)*
A field for representing data elements with both a value and a unit.

Creating a ValueAndUnitField named ‘frequency’, for example, will (under the hood) create the fields:
- `frequency_value`, the value of the element
- `frequency_unit`, the units in which the value is measured

When describing instances of your model (either when defining a *transform output* or when referencing fields using the *Indivo Query API*), you must refer to these field names, not the original *frequency* field name.

class `indivo.fields.AddressField` *(Type)*
A field for representing a physical address.

Creating an AddressField named ‘address’, for example, will (under the hood) create the fields:
- `address_country`, the country in which the address is located
- `address_city`, the city in which the address is located
- `address_postalcode`, the postalcode of the address
- `address_region`, the region (state, in the US) in which the address is located
- `address_street`, the street address (including street number, apartment number, etc.) at which the address is located

When describing instances of your model (either when defining a *transform output* or when referencing fields using the *Indivo Query API*), you must refer to these field names, not the original *address* field name.

class `indivo.fields.NameField` *(Type)*
A field for representing a person’s name.

Creating a NameField named ‘name’, for example, will (under the hood) create the fields:
- `name_family`, the family (last) name of the person
- `name_given`, the given (first) name of the person
- `name_middle`, the middle name of the person
- `name_prefix`, the prefix (i.e. ‘Mr.’, ‘Sir’, etc.) for the person’s name
- `name_suffix`, the suffix (i.e. ‘Jr.’, ‘Ph.D.’, etc.) for the person’s name

When describing instances of your model (either when defining a *transform output* or when referencing fields using the *Indivo Query API*), you must refer to these field names, not the original *name* field name.

class `indivo.fields.TelephoneField` *(Type)*
A field for representing a telephone number.

Creating a TelephoneField named ‘phone’, for example, will (under the hood) create the fields:
• phone_type, The type of the phone number, limited to h (home), w (work), or c (cell)
• phone_number, The actual phone number
• phone_preferred_p, Whether or not this number is a preferred method of contact (True or False)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original phone field name.

class indivo.fields.PharmacyField

A field for representing a pharmacy.

Creating a PharmacyField named ‘pharmacy’, for example, will (under the hood) create three fields:

• pharmacy_ncpdpid, the pharmacy’s National Council for Prescription Drug Programs (NCPDP) ID number
• pharmacy_adr, the address at which the pharmacy is located (an AddressField)
• pharmacy_org, the name of the organization that owns the pharmacy

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original pharmacy field name.

class indivo.fields.ProviderField

A field for representing a medical provider.

Creating a ProviderField named ‘doc’, for example, will (under the hood) create the fields:

• doc_dea_number, the provider’s Drug Enforcement Agency (DEA) number
• doc_ethnicity, the provider’s ethnicity
• doc_npi_number, the provider’s National Provider Identification (NPI) number
• doc_preferred_language, the provider’s preferred language
• doc_race, the provider’s race
• doc_adr, the provider’s address (an AddressField)
• doc_bday, the provider’s birth date
• doc_email, the provider’s email address
• doc_name, the provider’s name (a NameField)
• doc_tel_1, the provider’s primary phone number (a TelephoneField)
• doc_tel_2, the provider’s secondary phone number (a TelephoneField)
• doc_gender, the provider’s gender, limited to m (male) or f (female)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original doc field name.

class indivo.fields.VitalSignField

A field for representing a single measurement of a vital sign.

Creating a VitalSignField named ‘bp’, for example, will (under the hood) create the fields:

• bp_unit, the unit of the measurement
• bp_value, the value of the measurement
• bp_name, the name of the measurement (a CodedValueField)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original bp field name.
class indivo.fields.BloodPressureField (Type)
    A field for representing a blood pressure measurement.

    Creating a BloodPressureField named ‘bp’, for example, will (under the hood) create the fields:
    • bp_position, the position in which the measurement was taken (a CodedValueField)
    • bp_site, the site on the body at which the measurement was taken (a CodedValueField)
    • bp_method, the method of the measurement (a CodedValueField)
    • bp_diastolic, the diastolic blood pressure (a VitalSignField)
    • bp_systolic, the systolic blood pressure (a VitalSignField)

    When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original bp field name.

class indivo.fields.ValueRangeField (Type)
    A field for representing a range of values.

    Creating a ValueRangeField named ‘normal_range’, for example, will (under the hood) create the fields:
    • normal_range_max, the maximum value of the range (a ValueAndUnitField)
    • normal_range_min, the minimum value of the range (a ValueAndUnitField)

    When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original normal_range field name.

class indivo.fields.QuantitativeResultField (Type)
    A field for representing a quantitative result, and expected ranges for that result.

    Creating a QuantitativeResultField named ‘lab_result’, for example, will (under the hood) create the fields:
    • lab_result_non_critical_range, the range outside of which results are ‘critical’ (a ValueRangeField)
    • lab_result_normal_range, the range outside of which results are ‘abnormal’ (a ValueRangeField)
    • lab_result_value, the actual result (a ValueAndUnitField)

    When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original lab_result field name.

Simple Data Modeling Language (SDML)

For those less python-savvy who are still capable of thinking in terms of ‘fields’ and ‘types’ (which should be most people), we’ve defined a JSON-based modeling language for defining the very simple data models easily. SDML is less flexible than Django’s modeling language, but is much quicker to get started with and is less verbose for describing simple models. See our documentation of the language here.

Feeling Lost?

For help getting started, see our core data models, below, each of which provide definitions both in SDML and Django Model classes.
2.5.3 Data Models and the Query API

Since the *Query API* allows app developers to directly apply filters and ranges to the datamodels they are selecting, they need to know what fields they are allowed to query against. The answer is simple:

**ANY FIELD ON A DATA MODEL THAT IS NOT A RELATION TO ANOTHER MODEL MAY BE USED IN THE QUERY API!**

For example, we introduced the ‘Problem’ model above, which has the fields:

- *date_onset*: Date
- *date_resolution*: Date
- *name*: String
- *comments*: String
- *diagnosed_by*: String

If you were making an API call such as `GET /records/{RECORD_ID}/reports/minimal/problems/`, you could filter by any of:

- *date_onset*
- *date_resolution*
- *name*
- *comments*
- *diagnosed_by*

If the problems model were a bit more complicated, and had another field:

- *prescribed_med*: Medication

You wouldn’t be able to filter by *prescribed_med*, since that field is a relation to another model.

The only exceptions to this rule are *custom Django Model Fields*. Such fields are translated into fields with other names, as described above. Any of these fields may be used in the query API, but (for example), when looking at a model with a CodedValue element such as:

- *problem_type*: CodedValue

You would be able to filter by *problem_type_identifier*, *problem_type_title*, or *problem_type_system*, but not by *problem_type* itself.

2.5.4 Core Data Models

Here is a listing of the data models currently supported by Indivo. Each instance might define other, contributed models: see *below* for information on how to add data models to Indivo.

**Indivo Data Model: Allergy**

**Model Definition**

As *SDML*:
[{
    "__modelname__": "Allergy",
    "allergic_reaction": "CodedValue",
    "category": "CodedValue",
    "drug_allergen": "CodedValue",
    "drug_class_allergen": "CodedValue",
    "food_allergen": "CodedValue",
    "severity": "CodedValue"
},
{
    "__modelname__": "AllergyExclusion",
    "name": "CodedValue"
}]

As a Django Model Class:

```python
from indivo.models import Fact
from indivo.fields import CodedValueField

class Allergy(Fact):
    allergic_reaction = CodedValueField()
    category = CodedValueField()
    drug_allergen = CodedValueField()
    drug_class_allergen = CodedValueField()
    food_allergen = CodedValueField()
    severity = CodedValueField()

class AllergyExclusion(Fact):
    name = CodedValueField()
```

Examples

As SDMJ:

```json
[{
    "__modelname__": "Allergy",
    "allergic_reaction_title": "Anaphylaxis",
    "allergic_reaction_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
    "allergic_reaction_identifier": "39579001",
    "category_title": "Drug allergy",
    "category_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
    "category_identifier": "416098002",
    "drug_class_allergen_title": "Sulfonamide Antibacterial",
    "drug_class_allergen_system": "http://purl.bioontology.org/ontology/NDFRT/",
    "drug_class_allergen_identifier": "N0000175503",
    "severity_title": "Severe",
    "severity_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
    "severity_identifier": "24484000"
},
{
    "__modelname__": "AllergyExclusion",
    "name_title": "No known allergies",
    "name_identifier": "160244002",
    "name_system": "http://purl.bioontology.org/ontology/SNOMEDCT"
}]
```

As SDMX:
As a Fact object:

```python
from indivo.models import Allergy

allergy_fact = Allergy(
    allergic_reaction_title="Anaphylaxis",
    allergic_reaction_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    allergic_reaction_identifier="39579001",
    category_title="Drug allergy",
    category_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    category_identifier="416098002",
    drug_class_allergen_title="Sulfonamide Antibacterial",
    drug_class_allergen_system="http://purl.bioontology.org/ontology/NDFRT/",
    drug_class_allergen_identifier="N0000175503",
    severity_title="Severe",
    severity_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    severity_identifier="24484000",
)

allergy_exclusion = AllergyExclusion(
    name_title="No known allergies",
    name_identifier="160244002",
    name_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
)
```

**Indivo Data Model: Equipment**

**Model Definition**

As SDML:

```json
{
    "__modelname__": "Equipment",
    "date_started": "Date",
    "date_stopped": "Date",
```
"name": "String",
"vendor": "String",
"description": "String"
}

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class Equipment(Fact):
    date_started = models.DateField(null=True)
    date_stopped = models.DateField(null=True)
    name = models.CharField(max_length=40)
    vendor = models.CharField(max_length=40, null=True)
    description = models.TextField(null=True)
```

Examples

As SDMJ:

```
{
    "__modelname__": "Equipment",
    "date_started": "2009-02-05",
    "date_stopped": "2009-06-12",
    "name": "Pacemaker",
    "vendor": "Acme Medical Devices",
    "description": "it works!"
}
```

As SDMX:

```
<Models>
    <Model name="Equipment">
        <Field name="date_started">2009-02-05</Field>
        <Field name="date_stopped">2009-06-12</Field>
        <Field name="name">Pacemaker</Field>
        <Field name="vendor">Acme Medical Devices</Field>
        <Field name="description">it works!</Field>
    </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import Equipment
from indivo.lib.iso8601 import parse_utc_date as date

equipment_fact = Equipment(
    date_started=date("2009-02-05"),
    date_stopped=date("2009-06-12"),
    name="Pacemaker",
    vendor="Acme Medical Devices",
    description="it works!"
)
```
Indivo Data Model: Immunization

Model Definition

As SDML:

```json
{
    "__modelname__": "Immunization",
    "date": "Date",
    "administration_status": "CodedValue",
    "product_class": "CodedValue",
    "product_class_2": "CodedValue",
    "product_name": "CodedValue",
    "refusal_reason": "CodedValue"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField

class Immunization(Fact):
    date = models.DateTimeField(null=True)
    administration_status = CodedValueField()
    product_class = CodedValueField()
    product_class_2 = CodedValueField()
    product_name = CodedValueField()
    refusal_reason = CodedValueField()
```

Examples

As SDMJ:

```json
{
    "__modelname__": "Immunization",
    "date": "2009-05-16T12:00:00Z",
    "administration_status_title": "Not Administered",
    "administration_status_system": "http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#",
    "administration_status_identifier": "notAdministered",
    "product_class_title": "TYPHOID",
    "product_class_system": "http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#",
    "product_class_identifier": "TYPHOID",
    "product_name_title": "typhoid, oral",
    "product_name_system": "http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx#",
    "product_name_identifier": "25",
    "refusal_reason_title": "Allergy to vaccine/vaccine components, or allergy to eggs",
    "refusal_reason_system": "http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#",
    "refusal_reason_identifier": "allergy"
}
```

As SDMX:

```xml
<Models>
  <Model name="Immunization">
    <Field name="date">2009-05-16T12:00:00Z</Field>
    <Field name="administration_status_title">Not Administered</Field>
    <Field name="product_class_system">http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#</Field>
    <Field name="product_class_identifier">TYPHOID</Field>
    <Field name="product_name_system">http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx#</Field>
    <Field name="product_name_identifier">25</Field>
    <Field name="refusal_reason_system">http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#</Field>
    <Field name="refusal_reason_identifier">allergy</Field>
  </Model>
</Models>
```
As a Fact object:

```python
from indivo.models import Immunization
from indivo.lib.iso8601 import parse_utc_date as date

immunization_fact = Immunization(
    date=date("2009-05-16T12:00:00Z"),
    administration_status_title="Not Administered",
    administration_status_system="http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#",
    administration_status_identifier="notAdministered",
    product_class_title="TYPHOID",
    product_class_system="http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#",
    product_class_identifier="TYPHOID",
    product_name_title="typhoid, oral",
    product_name_system="http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx#",
    product_name_identifier="25",
    refusal_reason_title="Allergy to vaccine/vaccine components, or allergy to eggs",
    refusal_reason_system="http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#",
    refusal_reason_identifier="allergy",
)
```

Indivo Data Model: LabResult

Model Definition

As SDML:

```json
{
    "__modelname__": "LabResult",
    "abnormal_interpretation": "CodedValue",
    "accession_number": "String",
    "test_name": "CodedValue",
    "status": "CodedValue",
    "narrative_result": "String",
    "notes": "String",
    "quantitative_result": "QuantitativeResult",
    "collected_at": "Date",
    "collected_by_org": "Organization",
    "collected_by_name": "Name",
    "collected_by_role": "String"
}
```

As a Django Model Class:
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField, QuantitativeResultField, OrganizationField, NameField

class LabResult(Fact):
    abnormal_interpretation = CodedValueField()
    accession_number = models.CharField(max_length=255, null=True)
    test_name = CodedValueField()
    status = CodedValueField()
    narrative_result = models.CharField(max_length=255, null=True)
    notes = models.CharField(max_length=600, null=True)
    quantitative_result = QuantitativeResultField()
    collected_at = models.DateTimeField(null=True)
    collected_by_org = OrganizationField()
    collected_by_name = NameField()
    collected_by_role = models.CharField(max_length=255, null=True)

Examples

As SDMJ:

"__modelname__": "LabResult",
"abnormal_interpretation_title": "Normal",
"abnormal_interpretation_system": "http://smartplatforms.org/terms/codes/LabResultInterpretation#",
"abnormal_interpretation_identifier": "normal",

"accession_number": "AC09205823577",

"test_name_title": "Serum Sodium",
"test_name_system": "http://purl.bioontology.org/ontology/LNC/",
"test_name_identifier": "2951-2",

"status_title": "Final results": complete and verified",
"status_system": "http://smartplatforms.org/terms/codes/LabStatus#",
"status_identifier": "final",

"notes": "Blood sample appears to have hemolyzed",

"quantitative_result_non_critical_range_max_value": "155",
"quantitative_result_non_critical_range_max_unit": "mEq/L",
"quantitative_result_non_critical_range_min_value": "120",
"quantitative_result_non_critical_range_min_unit": "mEq/L",

"quantitative_result_normal_range_max_value": "145",
"quantitative_result_normal_range_max_unit": "mEq/L",
"quantitative_result_normal_range_min_value": "135",
"quantitative_result_normal_range_min_unit": "mEq/L",

"quantitative_result_value_value": "140",
"quantitative_result_value_unit": "mEq/L",

"collected_at": "2010-12-27T17:00:00Z",

"collected_by_org_name": "City Lab",
"collected_by_org_adr_country": "USA",
"collected_by_org_adr_city": "Springfield",
As SDMX:

```xml
<Models>
  <Model name="LabResult">
    <Field name="abnormal_interpretation_title">Normal</Field>
    <Field name="abnormal_interpretation_system">http://smartplatforms.org/terms/codes/LabResultInterpretation#</Field>
    <Field name="abnormal_interpretation_identifier">normal</Field>

    <Field name="accession_number">AC09205823577</Field>

    <Field name="test_name_title">Serum Sodium</Field>
    <Field name="test_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
    <Field name="test_name_identifier">2951-2</Field>

    <Field name="status_title">Final results: complete and verified</Field>
    <Field name="status_system">http://smartplatforms.org/terms/codes/LabStatus#</Field>
    <Field name="status_identifier">final</Field>

    <Field name="notes">Blood sample appears to have hemolyzed</Field>

    <Field name="quantitative_result_non_critical_range_max_value">155</Field>
    <Field name="quantitative_result_non_critical_range_max_unit">mEq/L</Field>
    <Field name="quantitative_result_non_critical_range_min_value">120</Field>
    <Field name="quantitative_result_non_critical_range_min_unit">mEq/L</Field>

    <Field name="quantitative_result_normal_range_max_value">145</Field>
    <Field name="quantitative_result_normal_range_max_unit">mEq/L</Field>
    <Field name="quantitative_result_normal_range_min_value">135</Field>
    <Field name="quantitative_result_normal_range_min_unit">mEq/L</Field>

    <Field name="quantitative_result_value_value">140</Field>
    <Field name="quantitative_result_value_unit">mEq/L</Field>

    <Field name="collected_at">2010-12-27T17:00:00Z</Field>
    <Field name="collected_by_org_name">City Lab</Field>
    <Field name="collected_by_org_adr_country">USA</Field>
    <Field name="collected_by_org_adr_city">Springfield</Field>
    <Field name="collected_by_org_adr_postalcode">11111</Field>
    <Field name="collected_by_org_adr_region">MA</Field>
    <Field name="collected_by_org_adr_street">20 Elm St</Field>

    <Field name="collected_by_name_family">Finnialispi</Field>
    <Field name="collected_by_name_given">Tad</Field>

    <Field name="collected_by_role">Lab Specialist</Field>
  </Model>
</Models>
```
As a Fact object:

```python
from indivo.models import LabResult
from indivo.lib.iso8601 import parse_utc_date as date

lab_fact = LabResult(
    abnormal_interpretation_title="Normal",
    abnormal_interpretation_system="http://smartplatforms.org/terms/codes/LabResultInterpretation#",
    abnormal_interpretation_identifier="normal",
    accession_number="AC09205823577",
    test_name_title="Serum Sodium",
    test_name_system="http://purl.bioontology.org/ontology/LNC/",
    test_name_identifier="2951-2",
    status_title="Final results: complete and verified",
    status_system="http://smartplatforms.org/terms/codes/LabStatus#",
    status_identifier="final",
    notes="Blood sample appears to have hemolyzed",
    quantitative_result_non_critical_range_max_value="155",
    quantitative_result_non_critical_range_max_unit="mEq/L",
    quantitative_result_non_critical_range_min_value="120",
    quantitative_result_non_critical_range_min_unit="mEq/L",
    quantitative_result_normal_range_max_value="145",
    quantitative_result_normal_range_max_unit="mEq/L",
    quantitative_result_normal_range_min_value="135",
    quantitative_result_normal_range_min_unit="mEq/L",
    quantitative_result_value_value="140",
    quantitative_result_value_unit="mEq/L",
    collected_at=date("2010-12-27T17:00:00Z"),
    collected_by_org_name="City Lab",
    collected_by_org_adr_country="USA",
    collected_by_org_adr_city="Springfield",
    collected_by_org_adr_postalcode="11111",
    collected_by_org_adr_region="MA",
    collected_by_org_adr_street="20 Elm St",
    collected_by_name_family="Finnialispi",
    collected_by_name_given="Tad",
    collected_by_role="Lab Specialist",
)
```

**Indivo Data Model: Medication**

**Model Definition**

As SDML:
["__modelname__": "Medication",
"drugName": "CodedValue",
"endDate": "Date",
"frequency": "ValueAndUnit",
"instructions": "String",
"provenance": "CodedValue",
"quantity": "ValueAndUnit",
"startDate": "Date",
"fulfillments": [{
   "__modelname__": "Fill",
   "date": "Date",
   "dispenseDaysSupply": "Number",
   "pbm": "String",
   "pharmacy": "Pharmacy",
   "provider": "Provider",
   "quantityDispensed": "ValueAndUnit"
}]
]

Note: Since SDML doesn’t provide for Boolean Fields, we are unable to define the dispense_as_written field properly in SDML. Our actual implementation of the Medication data model uses a Django Model Class for this reason.

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField, ValueAndUnitField, PharmacyField, ProviderField

class Medication(Fact):
    drugName = CodedValueField()
    endDate = models.DateField(null=True)
    frequency = ValueAndUnitField()
    instructions = models.CharField(max_length=255, null=True)
    provenance = CodedValueField()
    quantity = ValueAndUnitField()
    startDate = models.DateField(null=True)

class Fill(Fact):
    date = models.DateTimeField(null=True)
    dispenseDaysSupply = models.FloatField(null=True)
    pbm = models.CharField(max_length=255, null=True)
    pharmacy = PharmacyField()
    provider = ProviderField()
    quantityDispensed = ValueAndUnitField()
    medication = models.ForeignKey(Medication, null=True, related_name='fulfillments')

Examples

As SDMJ:

```
"endDate": "2007-08-14",
"frequency_value": "2",
"frequency_unit": "/d",
"instructions": "Take two tablets twice daily as needed for pain",
"provenance_title": "Derived by prescription",
"provenance_system": "http://smartplatforms.org/terms/codes/MedicationProvenance#",
"provenance_identifier": "prescription",
"quantity_value": "2",
"quantity_unit": "{tablet}",
"startDate": "2007-03-14",
"fulfillments": [
{
"__modelname__": "Fill",
"date": "2007-03-14T04:00:00Z",
"dispenseDaysSupply": "30",
"pbm": "T0000000001011",
"pharmacy_ncpdpid": "5235235",
"pharmacy_org": "CVS #588",
"pharmacy adr country": "Australia",
"pharmacy adr city": "WonderCity",
"pharmacy adr postalcode": "5555",
"pharmacy adr street": "111 Lake Drive",
"provider_dea_number": "325555555",
"provider_npi_number": "5235235",
"provider_email": "joshua.mandel@fake.emailserver.com",
"provider name given": "Josuha",
"provider name family": "Mandel",
"provider tel 1 type": "w",
"provider tel 1 number": "1-235-947-3452",
"quantityDispensed_value": "60",
"quantityDispensed_unit": "{tablet}"
},
{
"__modelname__": "Fill",
"date": "2007-04-14T04:00:00Z",
"dispenseDaysSupply": "30",
"pbm": "T0000000001011",
"pharmacy_ncpdpid": "5235235",
"pharmacy_org": "CVS #588",
"pharmacy adr country": "Australia",
"pharmacy adr city": "WonderCity",
"pharmacy adr postalcode": "5555",
"pharmacy adr street": "111 Lake Drive",
"provider_dea_number": "325555555",
"provider_npi_number": "5235235",
"provider_email": "joshua.mandel@fake.emailserver.com",
"provider name given": "Josuha",
"provider name family": "Mandel",
"provider tel 1 type": "w",
"provider tel 1 number": "1-235-947-3452",
"quantityDispensed_value": "60",
"quantityDispensed_unit": "{tablet}"
}]
}
As SDMX:

<Models>
  <Model name="Medication">
    <Field name="drugName_title">AMITRIPTYLINE HCL 50 MG TAB</Field>
    <Field name="drugName_system">http://purl.bioontology.org/ontology/RXNORM/</Field>
    <Field name="drugName_identifier">856845</Field>
    <Field name="endDate">2007-08-14</Field>
    <Field name="frequency_value">2</Field>
    <Field name="frequency_unit">/d</Field>
    <Field name="instructions">Take two tablets twice daily as needed for pain</Field>
    <Field name="provenance_title">Derived by prescription</Field>
    <Field name="provenance_system">http://smartplatforms.org/terms/codes/MedicationProvenance#</Field>
    <Field name="provenance_identifier">prescription</Field>
    <Field name="quantity_value">2</Field>
    <Field name="quantity_unit">{tablet}</Field>
    <Field name="startDate">2007-03-14</Field>
    <Field name="fulfillments">
      <Models>
        <Model name="Fill">
          <Field name="date">2007-03-14T04:00:00Z</Field>
          <Field name="dispenseDaysSupply">30</Field>
          <Field name="pbm">T00000000001011</Field>
          <Field name="pharmacy_ncpdpid">5235235</Field>
          <Field name="pharmacy_org">CVS #588</Field>
          <Field name="pharmacy_adr_country">Australia</Field>
          <Field name="pharmacy_adr_city">WonderCity</Field>
          <Field name="pharmacy_adr_postalcode">5555</Field>
          <Field name="pharmacy_adr_street">111 Lake Drive</Field>
          <Field name="provider_dea_number">325555555</Field>
          <Field name="provider_npi_number">5235235</Field>
          <Field name="provider_email">joshua.mandel@fake.emailserver.com</Field>
          <Field name="provider_name_given">Josuha</Field>
          <Field name="provider_name_family">Mandel</Field>
          <Field name="provider_tel_1_type">w</Field>
          <Field name="provider_tel_1_number">1-235-947-3452</Field>
          <Field name="provider_tel_1_preferred_p">true</Field>
          <Field name="quantityDispensed_value">60</Field>
          <Field name="quantityDispensed_unit">{tablet}</Field>
        </Model>
        <Model name="Fill">
          <Field name="date">2007-04-14T04:00:00Z</Field>
          <Field name="dispenseDaysSupply">30</Field>
          <Field name="pbm">T00000000001011</Field>
          <Field name="pharmacy_ncpdpid">5235235</Field>
          <Field name="pharmacy_org">CVS #588</Field>
          <Field name="pharmacy_adr_country">Australia</Field>
          <Field name="pharmacy_adr_city">WonderCity</Field>
          <Field name="pharmacy_adr_postalcode">5555</Field>
          <Field name="pharmacy_adr_street">111 Lake Drive</Field>
          <Field name="provider_dea_number">325555555</Field>
          <Field name="provider_npi_number">5235235</Field>
          <Field name="provider_email">joshua.mandel@fake.emailserver.com</Field>
          <Field name="provider_name_given">Josuha</Field>
          <Field name="provider_name_family">Mandel</Field>
          <Field name="provider_tel_1_type">w</Field>
          <Field name="provider_tel_1_number">1-235-947-3452</Field>
          <Field name="provider_tel_1_preferred_p">true</Field>
          <Field name="quantityDispensed_value">60</Field>
          <Field name="quantityDispensed_unit">{tablet}</Field>
        </Model>
      </Models>
  </Model>
</Models>
As a Fact object:

```python
from indivo.models import Medication, Fill
from indivo.lib.iso8601 import parse_utc_date as date

med = Medication(
    drugName_title="AMITRIPTYLINE HCL 50 MG TAB",
    drugName_system="http://purl.bioontology.org/ontology/RXNORM/",
    drugName_identifier="856845",
    endDate=date("2007-08-14"),
    frequency_value="2",
    frequency_unit="/d",
    instructions="Take two tablets twice daily as needed for pain",
    provenance_title="Derived by prescription",
    provenance_system="http://smartplatforms.org/terms/codes/MedicationProvenance#",
    provenance_identifier="prescription",
    quantity_value="2",
    quantity_unit="{tablet}",
    startDate=date("2007-03-14"),
)

fill1 = Fill(
    date=date("2007-03-14T04:00:00Z"),
    dispenseDaysSupply=30,
    pbm="T00000000001011",
    pharmacy_ncpdpid="5235235",
    pharmacy_org="CVS #588",
    pharmacy_adr_country="Australia",
    pharmacy_adr_city="WonderCity",
    pharmacy_adr_postalcode="5555",
    pharmacy_adr_street="111 Lake Drive",
    provider_dea_number="325555555",
    provider_npi_number="5235235",
    provider_email="joshua.mandel@fake.emailserver.com",
    provider_name_given="Josuha",
    provider_name_family="Mandel",
    provider_tel_1_type="w",
    provider_tel_1_number="1-235-947-3452",
    provider_tel_1_preferred_p=True,
    quantityDispensed_value="60",
    quantityDispensed_unit="{tablet}"
)

fill2 = Fill(
    date=date("2007-04-14T04:00:00Z"),
    dispenseDaysSupply=30,
    pbm="T00000000001011",
    pharmacy_ncpdpid="5235235",
    pharmacy_org="CVS #588",
    pharmacy_adr_country="Australia",
    pharmacy_adr_city="WonderCity",
    pharmacy_adr_postalcode="5555",
)```
 pharmacy_adr_street="111 Lake Drive",
 provider_dea_number="325555555",
 provider_npi_number="5235235",
 provider_email="joshua.mandel@fake.emailserver.com",
 provider_name_given="Joshua",
 provider_name_family="Mandel",
 provider_tel_1_type="w",
 provider_tel_1_number="1-235-947-3452",
 provider_tel_1_preferred_p=True,
 quantityDispensed_value="60",
 quantityDispensed_unit="/tablet",
)

# save the medication so we can relate other objects to it
med.save()
med.fulfillments = [fill1, fill2]
med.save()

**Indivo Data Model: Problem**

**Model Definition**

As *SDML*:

```
{
   "__modelname__": "Problem",
   "startDate": "Date",
   "endDate": "Date",
   "name": "CodedValue",
   "notes": "String"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from indivo.fields import CodedValueField
from django.db import models

class Problem(Fact):
    startDate = models.DateTimeField(null=True)
    endDate = models.DateTimeField(null=True)
    name = CodedValueField()
    notes = models.TextField(null=True)
```

**Examples**

As *SDMJ*:

```
{
   "__modelname__": "Problem",
   "startDate": "2009-05-16T12:00:00Z",
   "endDate": "2009-05-16T16:00:00Z",
   "name_title": "Backache (finding)",
   "name_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
   "name_identifier": "161891005"
}
```
As *SDMX*:

```xml
<Models>
  <Model name="Problem">
    <Field name="startDate">2009-05-16T12:00:00Z</Field>
    <Field name="endDate">2009-05-16T16:00:00Z</Field>
    <Field name="name_title">Backache (Finding)</Field>
    <Field name="name_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="name_identifier">161891005</Field>
  </Model>
</Models>
```

As a *Fact object*:

```python
from indivo.models import Problem
from indivo.lib.iso8601 import parse_utc_date as date

problem_fact = Problem(
    startDate=date("2009-05-16T12:00:00Z"),
    endDate=date("2009-05-16T16:00:00Z"),
    name_title="Backache (finding)",
    name_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    name_identifier="161891005",
)
```

**Indivo Data Model: Procedure**

**Model Definition**

As *SDML*:

```json
{
   "__modelname__": "Procedure",
   "date_performed": "Date",
   "name": "String",
   "name_type": "String",
   "name_value": "String",
   "name_abbrev": "String",
   "provider_name": "String",
   "provider_institution": "String",
   "location": "String",
   "comments": "String"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class Procedure(Fact):
    date_performed = models.DateTimeField(null=True)
    name = models.CharField(max_length=100)
    name_type = models.CharField(max_length=80, null=True)
    name_value = models.CharField(max_length=40, null=True)
    name_abbrev = models.CharField(max_length=20, null=True)
    provider_name = models.CharField(max_length=200, null=True)
    provider_institution = models.CharField(max_length=200, null=True)
```
location = models.CharField(max_length=100, null=True)
comments = models.TextField(null=True)

Examples

As SDMJ:

{
    "__modelname__": "Procedure",
    "date_performed": "2009-05-16T12:00:00",
    "name": "Appendectomy",
    "name_type": "http://codes.indivo.org/procedures#",
    "name_value": "123",
    "name_abbrev": "append",
    "provider_name": "Kenneth Mandl",
    "provider_institution": "Children’s Hospital Boston",
    "location": "300 Longwood Ave, Boston MA 02115",
    "comments": "Went great!"
}

As SDMX:

<Models>
    <Model name="Procedure">
        <Field name="date_performed">2009-05-16T12:00:00</Field>
        <Field name="name">Appendectomy</Field>
        <Field name="name_type">http://codes.indivo.org/procedures#</Field>
        <Field name="name_value">123</Field>
        <Field name="name_abbrev">append</Field>
        <Field name="provider_name">Kenneth Mandl</Field>
        <Field name="provider_institution">Children’s Hospital Boston</Field>
        <Field name="location">300 Longwood Ave, Boston MA 02115</Field>
        <Field name="comments">Went great!</Field>
    </Model>
</Models>

As a Fact object:

```python
from indivo.models import Procedure
from indivo.lib.iso8601 import parse_utc_date as date

procedure_fact = Procedure(
    date_performed=date("2009-05-16T12:00:00"),
    name="Appendectomy",
    name_type="http://codes.indivo.org/procedures#",
    name_value="123",
    name_abbrev="append",
    provider_name="Kenneth Mandl",
    provider_institution="Children’s Hospital Boston",
    location="300 Longwood Ave, Boston MA 02115",
    comments="Went great!"
)
```
Indivo Data Model: VitalSigns

Model Definition

As SDML:

```json
{
   "__modelname__": "VitalSigns",
   "date": "Date",
   "encounter": {
      "__modelname__": "Encounter",
      "startDate": "Date",
      "endDate": "Date",
      "facility": "Organization",
      "provider": "Provider",
      "encounterType": "CodedValue"
   },
   "bp": "BloodPressure",
   "bmi": "VitalSign",
   "heart_rate": "VitalSign",
   "height": "VitalSign",
   "oxygen_saturation": "VitalSign",
   "respiratory_rate": "VitalSign",
   "temperature": "VitalSign",
   "weight": "VitalSign"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import BloodPressureField, VitalSignField, CodedValueField, OrganizationField, ProviderField

class VitalSigns(Fact):
    date = models.DateTimeField(null=True)
    encounter = models.ForeignKey('Encounter', null=True)
    bp = BloodPressureField()
    bmi = VitalSignField()
    heart_rate = VitalSignField()
    height = VitalSignField()
    oxygen_saturation = VitalSignField()
    respiratory_rate = VitalSignField()
    temperature = VitalSignField()
    weight = VitalSignField()

class Encounter(Fact):
    startDate = models.DateTimeField(null=True)
    endDate = models.DateTimeField(null=True)
    facility = OrganizationField()
    provider = ProviderField()
    encounterType = CodedValueField()
```

Examples

As SDMJ:
{  
  "__modelname__": "VitalSigns",
  "date": "2009-05-16T12:00:00Z",
  "encounter": {
    "__modelname__": "Encounter",
    "startDate": "2009-05-16T12:00:00Z",
    "endDate": "2009-05-16T16:00:00Z",
    "facility_name": "Wonder Hospital",
    "facility_adr_country": "Australia",
    "facility_adr_city": "WonderCity",
    "facility_adr_postalcode": "5555",
    "facility_adr_street": "111 Lake Drive",
    "provider_dea_number": "3255555555",
    "provider_npi_number": "5235235",
    "provider_email": "joshua.mandel@fake.emailserver.com",
    "provider_name_given": "Josuha",
    "provider_name_family": "Mandel",
    "provider_tel_1_type": "w",
    "provider_tel_1_number": "1-235-947-3452",
    "provider_tel_1_preferred_p": true,
    "encounterType_title": "Ambulatory encounter",
    "encounterType_system": "http://smartplatforms.org/terms/codes/EncounterType#",
    "encounterType_identifier": "ambulatory"
  },
  "bp_position_title": "Sitting",
  "bp_position_identifier": "33586001",
  "bp_position_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
  "bp_site_title": "Right arm",
  "bp_site_identifier": "368209003",
  "bp_site_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
  "bp_method_title": "Auscultation",
  "bp_method_identifier": "auscultation",
  "bp_method_system": "http://smartplatforms.org/terms/codes/BloodPressureMethod#",
  "bp_diastolic_unit": "mm[Hg]",
  "bp_diastolic_value": 82,
  "bp_diastolic_name_title": "Intravascular diastolic",
  "bp_diastolic_name_identifier": "8462-4",
  "bp_diastolic_name_system": "http://purl.bioontology.org/ontology/LNC/",
  "bp_systolic_unit": "mm[Hg]",
  "bp_systolic_value": 132,
  "bp_systolic_name_title": "Intravascular systolic",
  "bp_systolic_name_identifier": "8480-6",
  "bp_systolic_name_system": "http://purl.bioontology.org/ontology/LNC/",
  "bmi_unit": "kg/m2",
  "bmi_value": 21.8,
  "bmi_name_title": "Body mass index",
  "bmi_name_system": "http://purl.bioontology.org/ontology/LNC/",
  "bmi_name_identifier": "39156-5",
  "heart_rate_unit": "(beats)/min",
  "heart_rate_value": 70,
  "heart_rate_name_title": "Heart rate",
  "heart_rate_name_system": "http://purl.bioontology.org/ontology/LNC/",
  "heart_rate_name_identifier": "8867-4",
  "height_unit": "m",
}
"height_value": 1.8,
"height_name_title": "Body height",
"height_name_system": "http://purl.bioontology.org/ontology/LNC/",
"height_name_identifier": "8302-2",

"oxygen_saturation_unit": "%{HemoglobinSaturation}'",
"oxygen_saturation_value": 99,
"oxygen_saturation_name_title": "Oxygen saturation",
"oxygen_saturation_name_system": "http://purl.bioontology.org/ontology/LNC/",
"oxygen_saturation_name_identifier": "2710-2",

"respiratory_rate_unit": "{breaths}/min",
"respiratory_rate_value": 16,
"respiratory_rate_name_title": "Respiration rate",
"respiratory_rate_name_system": "http://purl.bioontology.org/ontology/LNC/",
"respiratory_rate_name_identifier": "9279-1",

"temperature_unit": "Cel",
"temperature_value": 37,
"temperature_name_title": "Body temperature",
"temperature_name_system": "http://purl.bioontology.org/ontology/LNC/",
"temperature_name_identifier": "8310-5",

"weight_unit": "kg",
"weight_value": 70.8,
"weight_name_title": "Body weight",
"weight_name_system": "http://purl.bioontology.org/ontology/LNC/",
"weight_name_identifier": "3141-9"

As SDMX:

<Models>
  <Model name="VitalSigns">
    <Field name="date">2009-05-16T12:00:00Z</Field>
    <Field name="encounter">
      <Model name="Encounter">
        <Field name="startDate">2009-05-16T12:00:00Z</Field>
        <Field name="endDate">2009-05-16T16:00:00Z</Field>
        <Field name="facility_name">Wonder Hospital</Field>
        <Field name="facility_adr_country">Australia</Field>
        <Field name="facility_adr_city">WonderCity</Field>
        <Field name="facility_adr_postalcode">5555</Field>
        <Field name="facility_adr_street">111 Lake Drive</Field>
        <Field name="provider_dea_number">325555555</Field>
        <Field name="provider_npi_number">5235235</Field>
        <Field name="provider_email">joshua.mandel@fake.emailserver.com</Field>
        <Field name="provider_name_given">Josuha</Field>
        <Field name="provider_name_family">Mandel</Field>
        <Field name="provider_tel_1_type">w</Field>
        <Field name="provider_tel_1_number">1-235-947-3452</Field>
        <Field name="provider_tel_1_preferred_p">true</Field>
        <Field name="encounterType_title">Ambulatory encounter</Field>
        <Field name="encounterType_system">http://smartplatforms.org/terms/codes/EncounterType#</Field>
        <Field name="encounterType_identifier">ambulatory</Field>
      </Model>
    </Field>
    <Field name="bp_position_title">Sitting</Field>
  </Model>
</Models>
As a Fact object:

2.5. Indivo Data Models
from indivo.models import Encounter, VitalSigns
from indivo.lib.iso8601 import parse_utc_date as date

encounter_fact = Encounter(
    startDate=date("2009-05-16T12:00:00Z"),
    endDate=date("2009-05-16T16:00:00Z"),
    facility_name="Wonder Hospital",
    facility_adr_country="Australia",
    facility_adr_city="WonderCity",
    facility_adr_postalcode="5555",
    facility_adr_street="111 Lake Drive",
    provider_dea_number="325555555",
    provider_npi_number="5235235",
    provider_email="joshua.mandel@fake.emailserver.com",
    provider_name_given="Josuha",
    provider_name_family="Mandel",
    provider_tel_1_type="w",
    provider_tel_1_number="1-235-947-3452",
    provider_tel_1_preferred_p=True,
    encounterType_title="Ambulatory encounter",
    encounterType_system="http://smartplatforms.org/terms/codes/EncounterType#",
    encounterType_identifier="ambulatory",
)

encounter_fact.save()

# NOTE: all vitals readings are OPTIONAL. You don’t need
# to add all 56 fields here to create a VitalSigns object.
vitals_fact = VitalSigns(
    date=date("2009-05-16T12:00:00Z"),
    encounter=encounter_fact,

    # Blood Pressure
    bp_position_title="Sitting",
    bp_position_identifier="33586001",
    bp_position_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    bp_site_title="Right arm",
    bp_site_identifier="368209003",
    bp_site_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    bp_method_title="Auscultation",
    bp_method_identifier="auscultation",
    bp_method_system="http://smartplatforms.org/terms/codes/BloodPressureMethod#",
    bp_diastolic_unit="mm[Hg]",
    bp_diastolic_value=82,
    bp_diastolic_name_title="Intravascular diastolic",
    bp_diastolic_name_identifier="8462-4",
    bp_diastolic_name_system="http://purl.bioontology.org/ontology/LNC/",
    bp_systolic_unit="mm[Hg]",
    bp_systolic_value=132,
    bp_systolic_name_title="Intravascular systolic",
    bp_systolic_name_identifier="8480-6",
    bp_systolic_name_system="http://purl.bioontology.org/ontology/LNC/",

    # Body Mass Index
    bmi_unit="kg/m2",
    bmi_value=21.8,
    bmi_name_title="Body mass index",
    bmi_name_system="http://purl.bioontology.org/ontology/LNC/",
    bmi_name_identifier="39156-5",
)
# Heart Rate
heart_rate_unit="{beats}/min",
heart_rate_value=70,
heart_rate_name_title="Heart rate",
heart_rate_name_system="http://purl.bioontology.org/ontology/LNC/",
heart_rate_name_identifier="8867-4",

# Height
height_unit="m",
height_value=1.8,
height_name_title="Body height",
height_name_system="http://purl.bioontology.org/ontology/LNC/",
height_name_identifier="8302-2",

# Oxygen Saturation
oxygen_saturation_unit="%{HemoglobinSaturation}",
oxygen_saturation_value=99,
oxygen_saturation_name_title="Oxygen saturation",
oxygen_saturation_name_system="http://purl.bioontology.org/ontology/LNC/",
oxygen_saturation_name_identifier="2710-2",

# Respiratory Rate
respiratory_rate_unit="{breaths}/min",
respiratory_rate_value=16,
respiratory_rate_name_title="Respiration rate",
respiratory_rate_name_system="http://purl.bioontology.org/ontology/LNC/",
respiratory_rate_name_identifier="9279-1",

# Temperature
temperature_unit="Celsius",
temperature_value=37,
temperature_name_title="Body temperature",
temperature_name_system="http://purl.bioontology.org/ontology/LNC/",
temperature_name_identifier="8310-5",

# Weight
weight_unit="kg",
weight_value=70.8,
weight_name_title="Body weight",
weight_name_system="http://purl.bioontology.org/ontology/LNC/",
weight_name_identifier="3141-9",
}

**Indivo Data Model: Simple_Clinical_Note**

**Model Definition**

As `SDML`:

```
{
  "__modelname__": "SimpleClinicalNote",
  "date_of_visit": "Date",
  "finalized_at": "Date",
  "visit_type": "String",
  "visit_type_type": "String",
  "visit_type_value": "String",
```

2.5. Indivo Data Models
As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class SimpleClinicalNote(Fact):
    date_of_visit = models.DateTimeField()
    finalized_at = models.DateTimeField(null=True)
    visit_type = models.CharField(null=True, max_length=100)
    visit_type_type = models.CharField(max_length=80, null=True)
    visit_type_value = models.CharField(max_length=40, null=True)
    visit_type_abbrev = models.CharField(max_length=20, null=True)
    visit_location = models.CharField(max_length=200, null=True)
    specialty = models.CharField(null=True, max_length=100)
    specialty_type = models.CharField(max_length=80, null=True)
    specialty_value = models.CharField(max_length=40, null=True)
    specialty_abbrev = models.CharField(max_length=20, null=True)
    signed_at = models.DateTimeField(null=True)
    provider_name = models.CharField(null=True, max_length=200)
    provider_institution = models.CharField(max_length=200, null=True)
    chief_complaint = models.CharField(null=True, max_length=255)
    content = models.TextField(null=True)
```

Examples

As SDMJ:

```json
{
    "__modelname__": "SimpleClinicalNote",
    "date_of_visit": "2010-02-02T12:00:00Z",
    "finalized_at": "2010-02-03T13:12:00Z",
    "visit_type": "Acute Care",
    "visit_type_type": "http://codes.indivo.org/visit-types#",
    "visit_type_value": "123",
    "visit_type_abbrev": "acute",
    "visit_location": "Longfellow Medical",
    "specialty": "Hematology/Oncology",
    "specialty_type": "http://codes.indivo.org/specialties#",
    "specialty_value": "234",
    "specialty_abbrev": "hem-onc",
    "signed_at": "2010-02-03T13:12:00Z",
    "provider_name": "Kenneth Mandl",
    "provider_institution": "Children’s Hospital Boston",
    "chief_complaint": "stomach ache",
    "content": "String"
}
```
"content": "Patient presents with ..."
}

As SDMX:

<Models>
  <Model name="SimpleClinicalNote">
    <Field name="date_of_visit">2010-02-02T12:00:00Z</Field>
    <Field name="finalized_at">2010-03T13:12:00Z</Field>
    <Field name="visit_type">Acute Care</Field>
    <Field name="visit_type_type">http://codes.indivo.org/visit-types#</Field>
    <Field name="visit_type_value">123</Field>
    <Field name="visit_type_abbrev">acute</Field>
    <Field name="visit_location">Longfellow Medical</Field>
    <Field name="specialty">Hematology/Oncology</Field>
    <Field name="specialty_type">http://codes.indivo.org/specialties#</Field>
    <Field name="specialty_value">234</Field>
    <Field name="specialty_abbrev">hem-onc</Field>
    <Field name="signed_at">2010-03T13:12:00Z</Field>
    <Field name="provider_name">Kenneth Mandl</Field>
    <Field name="provider_institution">Children’s Hospital Boston</Field>
    <Field name="chief_complaint">stomach ache</Field>
    <Field name="content">Patient presents with ...</Field>
  </Model>
</Models>

As a Fact object:

from indivo.models import SimpleClinicalNote
from indivo.lib.iso8601 import parse_utc_date as date

simple_clinical_note_fact = SimpleClinicalNote(
    date_of_visit=date("2010-02-02T12:00:00Z"),
    finalized_at=date("2010-03T13:12:00Z"),
    visit_type="Acute Care",
    visit_type_type="http://codes.indivo.org/visit-types#",
    visit_type_value="123",
    visit_type_abbrev="acute",
    visit_location="Longfellow Medical",
    specialty="Hematology/Oncology",
    specialty_type="http://codes.indivo.org/specialties#",
    specialty_value="234",
    specialty_abbrev="hem-onc",
    signed_at=date("2010-03T13:12:00Z"),
    provider_name="Kenneth Mandl",
    provider_institution="Children’s Hospital Boston",
    chief_complaint="stomach ache",
    content="Patient presents with ..."
)

2.5.5 Advanced Data-Model Tasks

Adding Advanced Features to a Data-Model

For complicated data models, a simple SDML definition just won’t suffice. For a few specific features, such as custom object serialization or creation-time field validation, you can define (in python) an extra options file for a data model.
This file should be named `extra.py`, and can be dropped into the filesystem next to any data model, as described below. The file should contain subclasses of `indivo.data_models.options.DataModelOptions`, each of which describes the options for one data model defined in the `model.py` file in the same directory. Options are:

```python
class indivo.data_models.options.DataModelOptions (Type)
    Defines optional extra functionality for Indivo datamodels.
    
    To add options to a datamodel, subclass this class and override its attributes.
    
    Currently available options are:
    
    • `model_class_name`: Required. The name of the datamodel class to attach to.
    • `serializers`: Custom serializers for the data model. Should be set to a subclass of `indivo.serializers.DataModelSerializers`.
    • `field_validators`: Custom validators for fields on the data model. A dictionary, where keys are field names on the model, and values are lists of `Django Validators` to run against the field.
```

For example, here’s our options file for the Problem data model:

```python
from indivo.serializers import DataModelSerializers
from indivo.data_models.options import DataModelOptions
from indivo.validators import ExactValueValidator

SNOMED_URI = 'http://purl.bioontology.org/ontology/SNOMEDCT/

class ProblemSerializers(DataModelSerializers):
    def to_rdf(queryset, result_count, record=None, carenet=None):
        # ... our SMART RDF serializer implementation here ... #
        return 'some RDF'

class ProblemOptions(DataModelOptions):
    model_class_name = 'Problem'
    serializers = ProblemSerializers
    field_validators = {
        'name_system': [ExactValueValidator(SNOMED_URI)],
    }
```

Make sure to restart Indivo for your changes to take effect after you add your `extra.py` file—but there’s no need to reset Indivo.

**Adding Custom Serializers to a Data-Model**

By default, when returning data via the *generic reporting API*, Indivo will attempt to serialize data as *SDMJ* or *SDMX*, depending on the requested response format. If you need your data to come back in other formats, or if the default serializers aren’t smart enough to represent your data model correctly, you can implement custom serializers for the data model.

**Defining the Serializers**

Serializers for a data model are implemented as simple methods that take a Django queryset object, and return a serialized string. For a given data-model, you should define a subclass of `indivo.serializers.DataModelSerializers`, and add your desired serializers as methods on the class. Currently, available serializers are:
to_xml \(\text{queryset, result}_\text{count, record=None, carenet=None}\)
returns an XML string representing the model objects in \text{queryset}.

Parameters
- \text{queryset} (\text{QuerySet}) – the objects to serialize
- \text{result}_\text{count} (\text{integer}) – the total number of items in \text{queryset}
- \text{record} (\text{Record}) – the patient record that the objects belong to, if available.
- \text{carenet} (\text{Carenet}) – the Carenet via which the objects have been retrieved, if available.

Return type string

to_json \(\text{queryset, result}_\text{count, record=None, carenet=None}\)
returns a JSON string representing the model objects in \text{queryset}.

Parameters
- \text{queryset} (\text{QuerySet}) – the objects to serialize
- \text{result}_\text{count} (\text{integer}) – the total number of items in \text{queryset}
- \text{record} (\text{Record}) – the patient record that the objects belong to, if available.
- \text{carenet} (\text{Carenet}) – the Carenet via which the objects have been retrieved, if available.

Return type string

to_rdf \(\text{queryset, result}_\text{count, record=None, carenet=None}\)
returns an RDF/XML string representing the model objects in \text{queryset}.

Parameters
- \text{queryset} (\text{QuerySet}) – the objects to serialize
- \text{result}_\text{count} (\text{integer}) – the total number of items in \text{queryset}
- \text{record} (\text{Record}) – the patient record that the objects belong to, if available.
- \text{carenet} (\text{Carenet}) – the Carenet via which the objects have been retrieved, if available.

Return type string

For example, here’s a (non-functional) implementation of the serializers for the Problems data-model:

```
from indivo.serializers import DataModelSerializers

class ProblemSerializer(DataModelSerializers):
    def to_xml(queryset, result_count, record=None, carenet=None):
        return """<Problems>...bunch of problems here...</Problems>""

    def to_json(queryset, result_count, record=None, carenet=None):
        return """["Problem": "data here"], {"Problem": "More data here..."}]""

    def to_rdf(queryset, result_count, record=None, carenet=None):
        return """<rdf:RDF><rdf:Description rdf:type='indivo:Problem'>...RDF data here...</rdf:Description></rdf:RDF>""
```

A couple things to note:
- The to_*() methods DO NOT take \text{self} as their first argument. Under the hood, we actually rip the methods out of the serializers class and attach them directly to the data-model class.
- The \text{model_class_name} attribute is required, and indicates which data-model the serializers should be attached to.

2.5. Indivo Data Models 59
Liberaries for Serialization

When serializing models, the following libraries can come in handy:

- lxml.etree: Our favorite XML manipulation library. See http://lxml.de/tutorial.html for the details. Lxml is required for a running Indivo instance, so it will always be available for import (from lxml import etree).


- rdflib: Our favorite RDF manipulation library. See http://readthedocs.org/docs/rdflib/en/latest/. RDFLib may not be installed on all systems, so if you use it, make sure to install it first.

Attaching the Serializers to a Data Model

Adding custom serializers to a data-model is simple: simply set your `DataModelSerializers` subclass to the `serializers` attribute of a `DataModelOptions` subclass in an `extra.py` file (see above for info on adding advanced data-model options).

Adding Field Validation to a Data-Model

By default, data models defined in SDML are very permissive: all fields are nullable, and there are no constraints on valid data points other than their type (string, date, etc.). In some cases, a data element could satisfy these constraints, but still be invalid. For example, an Indivo Problem must have its name coded using SNOMED, so a problem without a snomed code is invalid.

Defining the Validators

In such cases, you can attach validators to the data model. Django Validators are essentially just python callables that raise a `django.core.exceptions.ValidationError` if they are called on an invalid data point. We’ve defined a couple of useful validators, though you could use any function you’d like.

For example, here’s a validator that will accept only the value 2:

```python
from django.core.exceptions import ValidationError

def validate_2(value):
    if value != 2:
        raise ValidationError("Invalid value: %s. Expected 2"%str(value))
```

Built in Validators

Django provides a number of built-in validators, for which a full reference exists here: https://docs.djangoproject.com/en/1.2/ref/validators/#built-in-validators.

In addition, Indivo defines a few useful validators in `indivo.validators`:

```python
class indivo.validators.ValueInSetValidator (valid_values, nullable=False)
    Validates that a value is within a set of possible values.

    The optional ‘nullable’ flag determines whether or not the value may also be empty.
```
class indivo.validators.ExactValueValidator(valid_value, nullable=False)

Validates that a value is exactly equal to a certain value.

The optional ‘nullable’ flag determines whether or not the value may also be empty.

Attaching Validators to a Data Model

Adding custom validators to a data-model is simple: simply add the validator to the field_validators attribute of a DataModelOptions subclass in an extra.py file (see above for info on adding advanced data-model options).

For example, let’s add the requirement that Problem names must be coded as snomed. We can write the validator using the built-in ExactValueValidator:

```python
from indivo.validators import ExactValueValidator
SNOMED_URI = 'http://purl.bioontology.org/ontology/SNOMEDCT/
snomed_validator = ExactValueValidator(SNOMED_URI)
```

We can then attach it to the name_system field of a Problem, which will guarantee that we only accept problems which identify themselves as having a snomed code for their names:

```python
class ProblemOptions(DataModelOptions):
    model_class_name = 'Problem'
    field_validators = {
        'name_system': [snomed_validator]
    }
```

Note that we put snomed_validator in a list, since we might theoretically add additional validators to the name_system field.

Adding Custom Data-Models to Indivo

As of version 1.1 of Indivo X, we’ve added a feature that makes it much easier to add (in a drag-and-drop fashion) new supported data models to an instance of Indivo. Adding a new data model to Indivo involves:

- Creating the data model definition
- Dropping the data model into the filesystem
- Migrating the database tables to support the new model

Defining the Data Model

As you saw above, data models can be defined in two formats: SDML or Django model classes. Simply produce a definition in one of the two forms, and save it to a file named model.sdml or model.py.

Dropping the Definition into the Filesystem

Indivo data models currently have the following layout on the filesystem:

```
indivo_server/
    indivo/
    ...
    data_models/
    core/
    allergy/
```
The `indivo/data_models/core/` directory contains all of our built-in data models, and you shouldn’t modify it. Since you are ‘contributing’ a data model to Indivo, add your data model to the `indivo/data_models/contrib/` directory. Simply:

- Create a new subdirectory under `indivo/data_models/contrib/`.
- Drop your model definition into that directory. This file MUST BE NAMED `MODEL.PY` OR `MODEL.SDML` to be identified as a data model.
- Add (optional) example files into that directory. Files should be named `example.sdmj`, `example.sdmx`, or `example.py`, and should be example instances of the data model as SDMJ, SDMX, or Fact objects respectively. If present, they will help others use and document your data model.
- Add an (optional) extras file to the directory. The file must be named `extra.py`, and may contain extra options for your data-model, such as custom serializers.
- Your final directory structure should now look something like:

  ```
  indivo_server/
  indivo/
  ...
  data_models/
  core/
  allergy/
  model.[sdml | py]
  example.[sdmj | sdmx | py]
  extra.py
  ...
  contrib/
  your_data_model/
  model.[sdml | py]
  example.[sdmj | sdmx | py]
  extra.py
  ```

### Migrating the Database

Indivo relies on the South migration tool to get the database synced with the latest data models. Once you’ve dropped your data model into the filesystem, South should be able to detect the necessary changes.

To detect the new model and generate migrations for it, run (from the `indivo_server` directory):

```
python manage.py schemamigration indivo --auto
```

You should see output like:

```
+ Added model indivo.YOURMODELNAME
Created 0018_auto__add_model_YOURMODELNAME.py. You can now apply this migration with: ./manage.py migrate indivo
```

To do a quick sanity check that you aren’t about to blow away your database, run:

```
python manage.py migrate indivo --db-dry-run -v2
```
This should output the SQL that will be run. Make sure this looks reasonable, ESPECIALLY if you are running Indivo on Oracle, where the South tool is still in alpha. If the SQL looks reasonable, go ahead and run the migration, with:

```
python manage.py migrate indivo
```

And you’re all set!

**Next Steps**

Make sure to restart Indivo for your changes to take effect.

**See also:**

Now you’ve added a new data model to Indivo: Congratulations! It can be stored in the database and queried via the API.

But until you map a *Schema* to it, you won’t be able to actually add data to your new model. To learn more, see:

- **Data Pipeline**
- **Adding Custom Schemas to Indivo**
- **Adding Custom Transforms to Indivo**

## 2.6 Indivo Schemas

### 2.6.1 Introduction

*Schemas* in Indivo are used to describe valid formats in which data may enter Indivo. Right now, we use XSDs as schemas, since we accept input data only in XML form, but in the future we might extend this to include schemas for validating other formats of data (OWL for RDF, etc.). Note that schemas describe the format of input data only! For information on how data is processed and stored in Indivo, see *Data Pipeline*.

There are a number of XML standards for medical activities, ranging from the CCR summary to the highly detailed CCD. None of these are particularly well tuned to the needs of a PCHR, where an individual datum may come from a hospital data feed, or from patient-based data entry. The Indivo schemas are built to serve the specific PCHR needs. Importantly, the Indivo schemas *use standard coding systems* wherever possible. The schemas are also ready for new coding systems as they emerge, especially in the realm of personally-controlled medicine with simplified terminology.

### 2.6.2 What if I want to store data that doesn’t match an Indivo schema?

Indivo X is designed to accept documents that conform to any XML schema, such as CCR, and even documents that are not XML, i.e. PDFs, MPEG, etc....

XML documents that conform to the built-in schemas can be immediately transformed, via the Indivo X *Data Pipeline*, into individual datapoints, which can then be queried using the *Indivo Reporting API*. XML documents that conform to custom schemas are not processed, and therefore cannot be retrieved using the reporting API (though you can still access them with *API calls for retrieving unprocessed documents*), which will return them in their original XML form.

If you want to extend Indivo to enable querying over data input according to a new schema, see *Adding Custom Schemas to Indivo*.
2.6.3 Namespace and XML Types

All of the default Indivo X document schemas are in a single namespace:

http://indivo.org/vocab/xml/documents#

The use of the trailing # enables simple RDF-like concatenation of namespace prefix and suffix to generate a single type URL. For example, an SDMX document in the Indivo documents namespace will have as its type:

http://indivo.org/vocab/xml/documents#Models

2.6.4 Design Rationale for Inclusion vs. Relation

Indivo X brings the ability to relate documents to one another using metadata, rather than document payload. This is particularly important when the payload might not be under the user’s control, i.e. a CCR document. It can also be useful even in the design of new Indivo schemas.

One could imagine separation the prescription information from the medication information, having two documents related to one another rather than one bigger document. However, our design rationale for now is to keep medication and its prescription data in the same XML document because those two chunks of data are generated in the same event. If, at some point, Indivo stores prescription filling information, then it is likely that this information would be more appropriately stored in a separate, linked document.

2.6.5 Core Schemas

All schema files and sample instance documents are available at http://indivo.org/vocab/xml/. Note that these schemas are only the ones that come with Indivo by default. Each instance of Indivo might define additional, custom schemas that are not documented here. See Adding Custom Schemas to Indivo for instructions on how to add custom schemas to Indivo.

Metadata and Indivo Internal Data Structures

Indivo Document Metadata Schema

Schema:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- didn’t place this in the Indivo namespace because it’s not medical payload -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Principal">
    <xs:sequence>
      <xs:element name="fullname" type="xs:string" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
    <!-- e.g. "fhh@apps.indivo.org" -->
    <xs:attribute name="id" type="xs:string" use="required" />
    <!-- e.g. "userapp" or "account" or "adminapp" -->
    <xs:attribute name="type" type="xs:string" use="required" />
  </xs:complexType>

  <xs:complexType name="Relation">
    <!-- e.g. "fhh@indivo.org" -->
    <xs:attribute name="id" type="xs:string" use="required" />
    <!-- e.g. "userapp" or "account" or "adminapp" -->
    <xs:attribute name="type" type="xs:string" use="required" />
  </xs:complexType>
</xs:schema>
```

Chapter 2. General Interest
<xs:attribute name="count" type="xs:integer" use="required" />
</xs:complexType>

<xs:element name="Document">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="createdAt" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
      <xs:element name="creator" type="Principal" minOccurs="1" maxOccurs="1" />
      <!-- if suppressedAt is non-null, then suppressor should be present -->
      <xs:element name="suppressedAt" type="xs:dateTime" minOccurs="0" maxOccurs="1" />
      <xs:element name="suppressor" type="Principal" minOccurs="0" maxOccurs="1" />
      <xs:element name="replacedBy" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
      <xs:element name="replaces" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
      <xs:element name="original" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
      <xs:element name="latest" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required" />
          <xs:attribute name="createdAt" type="xs:dateTime" use="required" />
          <xs:attribute name="createdBy" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
      <xs:element name="label" type="xs:string" minOccurs="0" maxOccurs="1" />
      <xs:element name="status" type="xs:string" minOccurs="0" maxOccurs="1" />
      <xs:element name="nevershare" type="xs:boolean" minOccurs="0" maxOccurs="1" />
      <xs:element name="relatesTo" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="relation" type="Relation" minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="isRelatedFrom" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="relation" type="Relation" minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type=""" digest="7e9bc09276e0829374fd810f96ed98a544649703db3a9bc231550a0b0e5bcb1c" size="77"
<createdAt>2009-05-04T17:05:33</createdAt>
<creator id="steve@indivo.org" type="account">
  <fullname>Steve Zabak</fullname>
</creator>
<suppressedAt>2009-05-06T17:05:33</suppressedAt>
<suppressor id="steve@indivo.org" type="account">
  <fullname>Steve Zabak</fullname>
</suppressor>
<original id="14c81023-c84f-496d-8b8e-9438280441d3" />
<latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
  <label>HBA1C reading</label>
  <status>active</status>
  <nevershare>false</nevershare>
  <relatesTo>
    <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
  </relatesTo>
  <isRelatedFrom>
    <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
  </isRelatedFrom>
</latest>
</Document>

Indivo Account Schema

An Indivo Account represents a single user of the system, with their basic info and the ways in which they authenticate. It is separate from a record.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <element name="Account">
    <complexType>
      <sequence>
        <element name="secret" type="string" minOccurs="0" maxOccurs="1" />
        <element name="fullName" type="string" minOccurs="1" maxOccurs="1" />
        <element name="contactEmail" type="string" minOccurs="1" maxOccurs="1" />
        <element name="lastLoginAt" type="dateTime" minOccurs="1" maxOccurs="1" />
        <element name="totalLoginCount" type="integer" minOccurs="1" maxOccurs="1" />
        <element name="failedLoginCount" type="integer" minOccurs="1" maxOccurs="1" />
        <element name="state" type="string" minOccurs="1" maxOccurs="1" />
        <element name="lastStateChange" type="dateTime" minOccurs="1" maxOccurs="1" />
        <element name="authSystem" minOccurs="0" maxOccurs="unbounded">
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```
<complexType>
    <attribute name="name" type="string" use="required" />
    <attribute name="username" type="string" use="required" />
</complexType>
</element>

</sequence>
<attribute name="id" type="string" use="required" />
</complexType>
</element>
</schema>

Example:

<Account id="joeuser@indivo.example.org">
    <fullName>Joe User</fullName>
    <contactEmail>joeuser@gmail.com</contactEmail>
    <lastLoginAt>2010-05-04T15:34:23Z</lastLoginAt>
    <totalLoginCount>43</totalLoginCount>
    <failedLoginCount>0</failedLoginCount>
    <state>active</state>
    <lastStateChange>2009-04-03T13:12:12Z</lastStateChange>

    <authSystem name="password" username="joeuser" />
    <authSystem name="hospital_sso" username="Joe_User" />
</Account>

Indivo PHA Schema

Information describing a Personal Health App (User App). Can be wrapped into a set of Apps.

Schema:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">

    <xs:complexType name="App">
        <xs:sequence>
            <xs:element name="startURLTemplate" type="xs:string" minOccurs="0" maxOccurs="1" />
            <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="description" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="autonomous" type="xs:boolean" minOccurs="1" maxOccurs="1" />
            <xs:element name="autonomousReason" type="xs:string" minOccurs="0" maxOccurs="1" />
            <xs:element name="frameable" type="xs:boolean" minOccurs="0" maxOccurs="1" />
            <xs:element name="ui" type="xs:boolean" minOccurs="0" maxOccurs="1" />
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required" />
    </xs:complexType>

    <xs:element name="App" type="App" />

    <xs:element name="Apps">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="App" type="App" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
Example:

```xml
<App id="problems@apps.indivo.org">
  <startURLTemplate>http://problems.indivo.org/auth/start?record_id={record_id}&amp;carenet_id={carenet_id}</startURLTemplate>
  <name>Problem List</name>
  <description>Managing your problem list</description>
  <autonomous>false</autonomous>
  <frameable>true</frameable>
  <ui>true</ui>
</App>
```

Example of multiple apps:

```xml
<Apps>
  <App id="problems@apps.indivo.org">
    <startURLTemplate>http://problems.indivo.org/auth/start?record_id={record_id}&amp;carenet_id={carenet_id}</startURLTemplate>
    <name>Problem List</name>
    <description>Managing your problem list</description>
    <autonomous>false</autonomous>
    <frameable>true</frameable>
    <ui>true</ui>
  </App>
</Apps>
```

Indivo Audit Log Schema

As of Beta 3, the logs will be returned as Indivo Reports according to the Indivo Reporting Schema. Each report item will be of type `<AuditEntry>`, as defined below:

```xml
<x:schema xmlns:x="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="AuditEntry">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="BasicInfo" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="datetime" type="xs:dateTime" use="required" />
            <xs:attribute name="view_func" type="xs:string" use="required" />
            <xs:attribute name="request_successful" type="xs:boolean" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="PrincipalInfo" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="effective_principal" type="xs:string" use="required" />
            <xs:attribute name="proxied_principal" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="Resources" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="carenet_id" type="xs:string" use="required" />
            <xs:attribute name="record_id" type="xs:string" use="required" />
            <xs:attribute name="pha_id" type="xs:string" use="required" />
            <xs:attribute name="document_id" type="xs:string" use="required" />
            <xs:attribute name="external_id" type="xs:string" use="required" />
            <xs:attribute name="message_id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
</xs:schema>
```
<xs:complexType>
  
  <xs:element name="RequestInfo" minOccurs="1" maxOccurs="1">
    <xs:complexType>
      <xs:attribute name="req_url" type="xs:string" use="required" />
      <xs:attribute name="req_ip_address" type="xs:string" use="required" />
      <xs:attribute name="req_domain" type="xs:string" use="required" />
      <xs:attribute name="req_method" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="ResponseInfo" minOccurs="1" maxOccurs="1">
    <xs:complexType>
      <xs:attribute name="resp_code" type="xs:integer" use="required" />
    </xs:complexType>
  </xs:element>

</xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>

Example:

<?xml version="1.0" encoding="utf-8" ?>

<AuditEntry>
  <BasicInfo datetime="2011-04-27T17:32:23Z" view_func="get_document" request_successful="true" />
  <PrincipalInfo effective_principal="myapp@apps.indivohealth.org" proxied_principal="me@indivohealth.org" />
  <Resources carenet_id="" record_id="123" pha_id="" document_id="acd" external_id="" message_id="" />
  <RequestInfo req_url="/records/123/documents/acd/" req_ip_address="127.0.0.1" req_domain="localhost" req_method="GET" />
  <ResponseInfo resp_code="200" />
</AuditEntry>

Indivo Carenet Schema

A list of carenets is returned when a user/app wants to know how a document is shared. However, this same list of
carenets might be used in a different setting. Thus, the “mode” attribute is optional. It indicates whether sharing in
this carenet was done explicitly, or via some implicit auto-share rule.

Schema:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">

  <xs:element name="Carenet">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="id" type="xs:string" use="required" />
        <xs:element name="mode" type="xs:string" use="optional" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>

Indivo Document Status History Schema

When a document’s status changes (archived, etc..), its history of changes is documented and available in this schema.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="DocumentStatusHistory">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DocumentStatus" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="reason" type="xs:string" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
            <xs:attribute name="by" type="xs:string" use="required" />
            <xs:attribute name="at" type="xs:dateTime" use="required" />
            <xs:attribute name="status" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="document_id" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```xml
<DocumentStatusHistory document_id="456">
  <DocumentStatus by="joeuser@indivo.example.org" at="2010-09-03T12:45:12Z" status="archived">
    <reason>no longer relevant</reason>
  </DocumentStatus>
</DocumentStatusHistory>
```

Indivo Inbox Message Schema

Indivo messages, sent to accounts (sometimes via a record), are represented with the following schema.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:complexType name="Message">
    <xs:sequence>
      <xs:element name="sender" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="received_at" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
      <xs:element name="read_at" type="xs:dateTime" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
<xs:element name="subject" type="xs:string" minOccurs="1" maxOccurs="1" />  
<xs:element name="severity" type="xs:string" minOccurs="1" maxOccurs="1" />  
<xs:element name="record" minOccurs="0" maxOccurs="1">  
    <xs:complexType>  
        <xs:attribute name="id" type="xs:string" use="required" />  
    </xs:complexType>  
</xs:element>  
<xs:element name="attachment" minOccurs="0" maxOccurs="unbounded">  
    <xs:complexType>  
        <xs:attribute name="num" type="xs:integer" use="required" />  
        <xs:attribute name="type" type="xs:string" use="required" />  
        <xs:attribute name="size" type="xs:integer" use="required" />  
        <xs:attribute name="doc_id" type="xs:string" />  
    </xs:complexType>  
</xs:element>  
</xs:sequence>  
<xs:attribute name="id" type="xs:string" use="required" />  
</xs:complexType>  
<xs:element name="Message" type="Message" />  
<xs:element name="Messages">  
    <xs:complexType>  
        <xs:sequence>  
            <xs:element name="Message" type="Message" minOccurs="0" maxOccurs="unbounded" />  
        </xs:sequence>  
    </xs:complexType>  
</xs:element>  
</xs:element>  
</xs:sequence>  
<xs:attribute name="id" type="xs:string" use="required" />  
</xs:complexType>  
<xs:element name="Message" type="Message" />  
<xs:element name="Messages">  
    <xs:complexType>  
        <xs:sequence>  
            <xs:element name="Message" type="Message" minOccurs="0" maxOccurs="unbounded" />  
        </xs:sequence>  
    </xs:complexType>  
</xs:element>  
</xs:element>  
</xs:sequence>  
<xs:attribute name="id" type="xs:string" use="required" />  
</xs:complexType>  
<xs:element name="Message" type="Message" />  
<xs:element name="Messages">  
    <xs:complexType>  
        <xs:sequence>  
            <xs:element name="Message" type="Message" minOccurs="0" maxOccurs="unbounded" />  
        </xs:sequence>  
    </xs:complexType>  
</xs:element>  
</xs:element>  
</xs:schema>

Example:

<Message id="879">  
    <sender>doctor@example.indivo.org</sender>  
    <received_at>2010-09-04T14:12:12Z</received_at>  
    <read_at>2010-09-04T17:13:24Z</read_at>  
    <subject>your test results are looking good</subject>  
    <severity>normal</severity>  
    <record id="123" />  
    <attachment num="1" type="http://indivo.org/vocab/xml/documents#Lab" size="12546" />  
</Message>

Another Example:

<Message id="879">  
    <sender>doctor@example.indivo.org</sender>  
    <received_at>2010-09-04T14:12:12Z</received_at>  
    <subject>your test results are looking good</subject>  
    <severity>normal</severity>  
    <record id="123" />  
    <attachment num="1" type="http://indivo.org/vocab/xml/documents#Lab" size="12546" />  
</Message>
Indivo Notification Schema

The Indivo Healthfeed includes notifications, represented by this schema:

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="Notifications">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Notification">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="sender" type="xs:string" minOccurs="1" maxOccurs="1" />
              <xs:element name="received_at" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
              <xs:element name="content" type="xs:string" minOccurs="1" maxOccurs="1" />
              <xs:element name="record" minOccurs="1" maxOccurs="1">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:string" use="required" />
                  <xs:attribute name="label" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
              <xs:element name="document" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:string" use="required" />
                  <xs:attribute name="label" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

Example:

```xml
<Notifications>
  <Notification id="468">
    <sender>labs@apps.indivo.org</sender>
    <received_at>2010-09-03T15:12:12Z</received_at>
    <content>A new lab result has been delivered to your account</content>
    <record id="123" label="Joe User" />
    <document id="579" label="Lab Test 2" />
  </Notification>
</Notifications>
```

Indivo Permissions Schema

Coming Soon...
Indivo Record Schema

The basic info for an Indivo record. Some of the attributes are there for indicating sharing relationships.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="Record">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="demographics" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="document_id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="created" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="at" type="xs:dateTime" use="optional" />
            <xs:attribute name="by" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required" />
      <xs:attribute name="label" type="xs:string" use="required" />
      <xs:attribute name="shared" type="xs:boolean" use="optional" />
      <xs:attribute name="carenet_id" type="xs:string" use="optional" />
      <xs:attribute name="carenet_name" type="xs:string" use="optional" />
      <xs:attribute name="role_label" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```xml
<Record id="123" label="Joe User" shared="true">
  <demographics document_id="467" />
  <created at="2010-10-23T10:23:34Z" by="indivoconnector@apps.indivo.org" />
</Record>
```

Indivo Request Token Schema

The Indivo UI Server needs to manage request tokens for apps so that it can display the appropriate authorization screens. This schema makes use of the Indivo PHA Schema.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:include schemaLocation="../pha/pha.xsd" />
  <xs:element name="RequestToken">
    <xs:complexType>
      <xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required" />
        <xs:attribute name="label" type="xs:string" use="required" />
        <xs:attribute name="shared" type="xs:boolean" use="optional" />
        <xs:attribute name="carenet_id" type="xs:string" use="optional" />
        <xs:attribute name="carenet_name" type="xs:string" use="optional" />
        <xs:attribute name="role_label" type="xs:string" use="optional" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```xml
<RequestToken id="123" label="Joe User" shared="true">
  <demographics document_id="467" />
  <created at="2010-10-23T10:23:34Z" by="indivoconnector@apps.indivo.org" />
</RequestToken>
```
Example:

```xml
<RequestToken token="XYZ">
  <record id="123" />
  <carenet />
  <kind>new</kind>
  <App id="problems@apps.indivo.org">
    <name>Problem List</name>
    <description>Managing your list of problems</description>
    <autonomous>false</autonomous>
    <frameable>true</frameable>
    <ui>true</ui>
  </App>
</RequestToken>
```

**Indivo Coded Values**

A coded value is a value taken from a coding system. It consists of a reference to the coding system (a URL), the code value, and the human-readable string. When the coding system is not used but a manual value is entered, the coding system and coded value are absent, leaving only the human-readable string.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
  <xs:complexType name="CodedValue">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="type" type="xs:anyURI" use="optional" />
        <xs:attribute name="value" type="xs:string" use="optional" />
        <xs:attribute name="abbrev" type="xs:string" use="optional" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```
• When a document comes into Indivo, its coded values may be expanded (with abbreviation and element content) or not (just the code and coding system).

• We will encourage applications to provide expanded coded values, but this will not be required.

• Reports will provide abbreviations and full names for all relevant codes by looking up against Indivo-stored copies of the coding systems. Documents will not be modified from what the sources send us, to follow the principles of store exactly the original data source (that’s required because the documents might be digitally signed.)

• Reports can flag codes whose abbreviations and full names do not match the coding system data (but we always show by default what the document says, we trust the source, not the coding system.)

• We then need a code lookup API for viewing single documents.

• codes.indivo.org will provide an API for interpreting codes.

**Indivo Data Values Schema**

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
  <xs:complexType name="Range">
    <xs:sequence>
      <!-- a missing minimum means < max -->
      <xs:element name="minimum" type="xs:double" minOccurs="0" maxOccurs="1" />
      <!-- a missing maximum means > min -->
      <xs:element name="maximum" type="xs:double" minOccurs="0" maxOccurs="1" />
      <!-- technically this schema allows a range with neither min nor max, which doesn’t mean much, -->
      <xs:element name="unit" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <!-- an ordinal, i.e. "2+" is coded using textValue -->
  <xs:complexType name="ValueAndUnit">
    <xs:sequence>
      <xs:element name="value" type="xs:double" minOccurs="0" maxOccurs="1" />
      <xs:element name="textValue" type="xs:string" minOccurs="0" maxOccurs="1" />
      <xs:element name="unit" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <!-- a result is an abstract base, only subtypes can be used -->
  <xs:complexType name="Result" abstract="true">
    <xs:sequence>
      <!-- HL7 defines flag types -->
      <xs:element name="flag" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
<!-- untyped strings, e.g. "positive" -->
<xs:complexType name="ResultInSet">
  <xs:complexContent>
    <xs:extension base="indivo:Result">
      <xs:sequence>
        <xs:element name="value" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Concentration">
  <xs:complexContent>
    <xs:extension base="indivo:ValueOrRange" />
  </xs:complexContent>
</xs:complexType>

Indivo Provider Schema

A provider is typically an MD with an institution affiliation.
Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
   xmlns:indivo="http://indivo.org/vocab/xml/documents#"
   elementFormDefault="qualified">
  <complexType name="Provider">
    <sequence>
      <element name="name" type="string" minOccurs="1" maxOccurs="1" />  
      <element name="institution" type="string" minOccurs="1" maxOccurs="1" />
    </sequence>
  </complexType>

  <complexType name="LabProvider">
    <sequence>
      <element name="name" type="string" minOccurs="1" maxOccurs="1" />  
      <element name="address" type="string" minOccurs="1" maxOccurs="1" />
    </sequence>
  </complexType>

  <complexType name="Signature">
    <sequence>
      <element name="at" type="dateTime" minOccurs="0" maxOccurs="1" />
      <element name="provider" type="indivo:Provider" minOccurs="1" maxOccurs="1" />
    </sequence>
  </complexType>
</schema>
```

**Reporting**

**Indivo Reporting Schema**

See the schema for *Indivo Document Metadata Schema* and specific *indivo:doc* schemas.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
   elementFormDefault="qualified">
  <xs:complexType name="Report">
    <xs:sequence>
      <xs:element name="Meta" minOccurs="1" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Document" minOccurs="1" maxOccurs="1" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Item" minOccurs="1" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:choice minOccurs="1" maxOccurs="1">
              <xs:element name="Allergy" />
              <xs:element name="Equipment" />
              <xs:element name="Immunization" />
            </xs:choice>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
<xs:element name="LabReport" />
<xs:element name="Medication" />
<xs:element name="Problem" />
<xs:element name="Procedure" />
<xs:element name="SimpleClinicalNote" />
<xs:element name="VitalSign" />
<xs:element name="AggregateReport" />
</xs:choice>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<!--
<xs:element name="Reports">
<xs:complexType>
<xs:sequence>
<xs:element name="Summary" minOccurs="1" maxOccurs="1">
<xs:complexType>
<xs:attribute name="total_document_count" use="optional" type="xs:int" />
<xs:attribute name="limit" use="optional" type="xs:int" />
<xs:attribute name="offset" use="optional" type="xs:int" />
<xs:attribute name="order_by" use="optional" type="xs:string"/>
</xs:complexType>
</xs:element>
<xs:element name="QueryParams" minOccurs="1" maxOccurs="1">
<xs:complexType>
<xs:sequence>
<xs:element name="GroupBy" minOccurs="0" maxOccurs="1">
<xs:complexType>
<xs:attribute name="value" use="required" type="xs:string" />
</xs:complexType>
</xs:element>
<xs:element name="DateGroup" minOccurs="0" maxOccurs="1">
<xs:complexType>
<xs:attribute name="value" use="required" type="xs:string" />
</xs:complexType>
</xs:element>
<xs:element name="AggregateBy" minOccurs="0" maxOccurs="1">
<xs:complexType>
<xs:attribute name="value" use="required" type="xs:string" />
</xs:complexType>
</xs:element>
<xs:element name="DateRange" minOccurs="0" maxOccurs="1">
<xs:complexType>
<xs:attribute name="value" use="required" type="xs:string" />
</xs:complexType>
</xs:element>
<xs:element name="Filters" minOccurs="0" maxOccurs="1">
<xs:complexType>
<xs:sequence>
<xs:element name="Filter" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:attribute name="name" use="required" type="xs:string" />
<xs:attribute name="value" use="required" type="xs:string" />
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
-->
Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*" />
    <Filters>
      <Filter name="lab_type" value="hematology"/>
    </Filters>
  </QueryParams>
  <Report>
    <Meta>
      <Document id="261ca370-927f-41af-b001-7b615c7a468e" type="http://indivo.org/vocab/xml/documents#Lab" size="1653" digest="0799971784e5a2d199cd6585415a8cd57f7bf9e4f8c8f74ef67a1009a1481cd6">
        <createdAt>2011-05-02T17:48:13Z</createdAt>
        <creator id="mymail@mail.ma" type="Account">
          <fullname>full name</fullname>
        </creator>
        <original id="261ca370-927f-41af-b001-7b615c7a468e"/>
        <label>testing</label>
        <status>active</status>
        <nevershare>false</nevershare>
      </Document>
    </Meta>
    <Item>
      <LabReport xmlns="http://indivo.org/vocab/xml/documents#">
        <dateMeasured>1998-07-16T12:00:00Z</dateMeasured>
        <labType>hematology</labType>
        <laboratory>
          <name>Quest</name>
          <address>300 Longwood Ave, Boston MA 02215</address>
        </laboratory>
        <comments>was looking pretty sick</comments>
        <firstPanelName>CBC</firstPanelName>
      </LabReport>
    </Item>
  </Report>
  <Report>
    <Meta>
      <Document id="1b7270a6-5925-450c-9273-5a74386cef63" type="http://indivo.org/vocab/xml/documents#Lab">
        <createdAt>2011-05-02T17:48:13Z</createdAt>
        <creator id="mymail@mail.ma" type="Account">
          <fullname>full name</fullname>
        </creator>
        <original id="1b7270a6-5925-450c-9273-5a74386cef63"/>
        <label>testing</label>
        <status>active</status>
      </Document>
    </Meta>
  </Report>
</Reports>
```
Indivo Aggregate Report Schema

This schema describes report items returned in aggregate form.

Schema:

```xml
    elementFormDefault="qualified">

  <xs:complexType name="AggregateReport">
    <xs:attribute name="value" type="xs:string" use="required" />
    <xs:attribute name="group" type="xs:string" use="optional" />
  </xs:complexType>

</xs:schema>
```

Example:

```xml
```

Indivo Generic Aggregate Reports Schema

This schema describes report items returned in XML aggregate form from Generic Reports.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="AggregateReport">
    <xs:attribute name="value" type="xs:string" use="required" />
    <xs:attribute name="group" type="xs:string" use="optional" />
  </xs:complexType>

</xs:schema>
```
Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<AggregateReports>
    <AggregateReport value="1" group="2009-07" />
    <AggregateReport value="4" group="2009-08" />
</AggregateReports>
```

Special Documents

**Indivo Document Demographics Schema**

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:indivo="http://indivo.org/vocab/xml/documents#"
    elementFormDefault="qualified"
    targetNamespace="http://indivo.org/vocab/xml/documents#">

    <xs:complexType name="Name">
        <xs:sequence>
            <xs:element name="familyName" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="givenName" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="middleName" type="xs:string" minOccurs="0" maxOccurs="1" />
            <xs:element name="prefix" type="xs:string" minOccurs="0" maxOccurs="1" />
            <xs:element name="suffix" type="xs:string" minOccurs="0" maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="PhoneType">
        <xs:restriction base="xs:normalizedString">
            <xs:enumeration value="h"/>
            <xs:enumeration value="w"/>
            <xs:enumeration value="c"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="GenderType">
        <xs:restriction base="xs:normalizedString">
            <xs:enumeration value="female"/>
            <xs:enumeration value="male"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="Telephone">
        <xs:sequence>
        </xs:sequence>
    </xs:complexType>

```

2.6. Indivo Schemas
<xs:element name="type" type="indivo:PhoneType" minOccurs="0" maxOccurs="1" />
<xs:element name="number" type="xs:string" minOccurs="1" maxOccurs="1" />
<xs:element name="preferred" type="xs:boolean" minOccurs="0" maxOccurs="1" default="false" />
</xs:sequence>
</xs:complexType>

<xs:complexType name="Address">
<xs:sequence>
<xs:element name="country" type="xs:string" minOccurs="0" maxOccurs="1" />
<xs:element name="city" type="xs:string" minOccurs="0" maxOccurs="1" />
<xs:element name="postalCode" type="xs:string" minOccurs="0" maxOccurs="1" />
<xs:element name="region" type="xs:string" minOccurs="0" maxOccurs="1" />
<xs:element name="street" type="xs:string" minOccurs="0" maxOccurs="1" />
</xs:sequence>
</xs:complexType>

<xs:element name="Demographics">
<xs:complexType>
<xs:sequence>
<xs:element name="dateOfBirth" type="xs:date" minOccurs="1" />
<xs:element name="gender" type="indivo:GenderType" minOccurs="1" />
<xs:element name="email" type="xs:string" minOccurs="0" />
<xs:element name="ethnicity" type="xs:string" minOccurs="0" />
<xs:element name="preferredLanguage" type="xs:string" minOccurs="0" />
<xs:element name="race" type="xs:string" minOccurs="0" />
<xs:element name="Name" type="indivo:Name" minOccurs="1" />
<xs:element name="Telephone" type="indivo:Telephone" minOccurs="0" maxOccurs="2" />
<xs:element name="Address" type="indivo:Address" minOccurs="0" maxOccurs="1" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

Example:

<?xml version="1.0" encoding="utf-8" ?>
<Demographics xmlns="http://indivo.org/vocab/xml/documents#">
  <dateOfBirth>1939-11-15</dateOfBirth>
  <gender>male</gender>
  <email>test@fake.org</email>
  <ethnicity>Scottish</ethnicity>
  <preferredLanguage>EN</preferredLanguage>
  <race>caucasian</race>
  <Name>
    <familyName>Wayne</familyName>
    <givenName>Bruce</givenName>
    <middleName>Quentin</middleName>
    <prefix>Mr</prefix>
    <suffix>Jr</suffix>
  </Name>
  <Telephone>
    <type>h</type>
    <number>555-5555</number>
    <preferred>true</preferred>
  </Telephone>
  <Telephone>
    <type>c</type>
    <number>555-6666</number>
  </Telephone>
</Demographics>
Indivo Document Annotation Schema

Medical documents can be annotated in Indivo, with a document that’s added in relation to the annotated document. This is a simple schema for these text-based annotations.

Schema:
Coming Soon!

Example:
Coming Soon!

The relationship to the annotated document is maintained in Indivo metadata, and the annotation can, optionally, store the SHA256 hash of the referenced document for robustness.

Medical Documents

Indivo Document Schema: Procedure

A procedure is effectively a surgical event (though some are not exactly surgical.)

See also the schema for Indivo Coded Values and for Indivo Provider Schema.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://indivo.org/vocab/xml/documents#" elementFormDefault="qualified">
    <include schemaLocation="../../common/codes.xsd" />
    <include schemaLocation="../../common/provider.xsd" />

    <element name="Procedure">
        <complexType>
            <sequence>
                <element name="datePerformed" type="dateTime" minOccurs="0" maxOccurs="1" />
                <element name="name" type="indivo:CodedValue" minOccurs="1" maxOccurs="1" />
                <element name="provider" type="indivo:Provider" minOccurs="0" />
                <element name="location" type="string" minOccurs="0" maxOccurs="1" />
                <element name="comments" type="string" minOccurs="0" maxOccurs="1" />
            </sequence>
        </complexType>
    </element>
</schema>
```

Example:
<Procedure xmlns="http://indivo.org/vocab/xml/documents#">
  <datePerformed>2009-05-16T12:00:00</datePerformed>
  <name type="http://codes.indivo.org/procedures#" value="85" abbrev="append">Appendectomy</name>
  <provider>
    <name>Kenneth Mandl</name>
    <institution>Children's Hospital Boston</institution>
  </provider>
</Procedure>

Indivo Document Schema: Equipment

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://indivo.org/vocab/xml/documents#"
  xmlns:indivo="http://indivo.org/vocab/xml/documents#"
  elementFormDefault="qualified">
  <element name="Equipment">
    <complexType>
      <sequence>
        <element name="dateStarted" type="date" minOccurs="0" maxOccurs="1" />
        <element name="dateStopped" type="date" minOccurs="0" maxOccurs="1" />
        <element name="type" type="string" minOccurs="0" maxOccurs="1" />
        <element name="name" type="string" minOccurs="1" maxOccurs="1" />
        <element name="vendor" type="string" minOccurs="0" maxOccurs="1" />
        <element name="id" type="string" minOccurs="0" maxOccurs="1" />
        <element name="description" type="string" minOccurs="0" maxOccurs="1" />
        <element name="specification" type="string" minOccurs="0" maxOccurs="1" />
        <element name="certification" type="string" minOccurs="0" maxOccurs="1" />
      </sequence>
    </complexType>
  </element>
</schema>
```

Example:

```xml
<Equipment xmlns="http://indivo.org/vocab/xml/documents#">
  <dateStarted>2009-02-05</dateStarted>
  <dateStopped>2010-06-12</dateStopped>
  <type>cardiac</type>
  <name>Pacemaker</name>
  <vendor>Acme Medical Devices</vendor>
  <id>167-ABC-23</id>
  <description>it works</description>
  <specification>blah blah blah</specification>
</Equipment>
```

Indivo Document Schema: Simple Clinical Note

A full clinical note needs to contain a number of coded problems, etc. Some hospital systems do not have fully normalized clinical notes, in which case they can use this schema to store some simple attributes and the main free-form text of the note.

See also Indivo Coded Values and Indivo Provider Schema.
Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://indivo.org/vocab/xml/documents#" elementFormDefault="qualified">
  <include schemaLocation="../../common/codes.xsd" />
  <include schemaLocation="../../common/provider.xsd" />
  <!-- this is mostly a chunk o' text. More normalized clinical notes will be in a diff schema -->
  <element name="SimpleClinicalNote">
    <complexType>
      <sequence>
        <element name="dateOfVisit" type="dateTime" minOccurs="1" maxOccurs="1" />
        <element name="finalizedAt" type="dateTime" minOccurs="0" maxOccurs="1" />
        <element name="visitType" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
        <element name="visitLocation" type="string" minOccurs="0" maxOccurs="1" />
        <element name="specialty" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
        <element name="signature" type="indivo:Signature" minOccurs="1" maxOccurs="unbounded" />
        <element name="chiefComplaint" type="string" minOccurs="0" maxOccurs="1" />
        <element name="content" minOccurs="0" maxOccurs="1" type="string" />
      </sequence>
    </complexType>
  </element>
</schema>
```

Example:

```xml
<SimpleClinicalNote xmlns="http://indivo.org/vocab/xml/documents#">
  <dateOfVisit>2010-02-02T12:00:00Z</dateOfVisit>
  <finalizedAt>2010-02-03T13:12:00Z</finalizedAt>
  <visitType type="http://codes.indivo.org/visit-types#" value="acute">Acute Care</visitType>
  <visitLocation>Longfellow Medical</visitLocation>
  <specialty type="http://codes.indivo.org/specialties#" value="hem-onc">Hematology/Oncology</specialty>

  <signature>
    <at>2010-02-03T13:12:00Z</at>
    <provider>
      <name>Kenneth Mandl</name>
      <institution>Children's Hospital Boston</institution>
    </provider>
  </signature>

  <signature>
    <provider>
      <name>Isaac Kohane</name>
      <institution>Children's Hospital Boston</institution>
    </provider>
  </signature>

  <chiefComplaint>stomach ache</chiefComplaint>
  <content>
    Patient presents with ...
  </content>
</SimpleClinicalNote>
```
Indivo Document Schema: SDMX

For any data model in Indivo that can be represented in SDML, we will accept data in the form of SDMX.

Schema:

```xml
	namespace="http://indivo.org/vocab/xml/documents#">
  <complexType name="ModelType">
    <sequence>
      <element name="Field" minOccurs="0" maxOccurs="unbounded">
        <complexType mixed="true">
          <choice>
            <element name="Model" type="indivo:ModelType" minOccurs="0" maxOccurs="1"/>
            <element name="Models" type="indivo:ModelsType" minOccurs="0" maxOccurs="1"/>
          </choice>
          <attribute name="name" type="string" use="required"/>
        </complexType>
      </element>
    </sequence>
    <attribute name="name" type="string" use="required"/>
    <attribute name="documentId" type="string" use="optional"/>
  </complexType>

  <complexType name="ModelsType">
    <sequence>
      <element name="Model" type="indivo:ModelType" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <element name="Models" type="indivo:ModelsType"/>
</schema>
```

Example:

```xml
<Models xmlns="http://indivo.org/vocab/xml/documents#">
  <Model name="TestMedication" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
    <Field name="date_started">2010-10-01T00:00:00Z</Field>
    <Field name="name">ibuprofen</Field>
    <Field name="brand_name">Advil</Field>
    <Field name="date_stopped">2010-10-31T00:00:00Z</Field>
    <Field name="prescription">
      <Model name="TestPrescription" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
        <Field name="prescribed_by_name">Kenneth D. Mandl</Field>
        <Field name="prescribed_by_institution">Children’s Hospital Boston</Field>
        <Field name="prescribed_on">2010-09-30T00:00:00Z</Field>
        <Field name="prescribed_stop_on">2010-10-31T00:00:00Z</Field>
      </Model>
    </Field>
    <Field name="fills">
      <Models>
        <Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
          <Field name="date_filled">2010-10-01T00:00:00Z</Field>
          <Field name="supply_days">15</Field>
          <Field name="filled_at_name">CVS</Field>
        </Model>
        <Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
          <Field name="date_filled">2010-10-16T00:00:00Z</Field>
        </Model>
      </Models>
    </Field>
  </Model>
</Models>
```
2.6.6 Adding Custom Schemas to Indivo

As of version 1.1 of Indivo X, we’ve added a feature that makes it much easier to add (in a drag-and-drop fashion) new supported schemas to an instance of Indivo. Adding a new schema to Indivo involves:

- Creating the schema
- Mapping the schema to Indivo’s Data Models
- Dropping the schema into the filesystem

Creating the Schema

Indivo currently accepts schemas only in XSD form. There are numerous tutorials and tools on the web to help you create an XSD, so we won’t presume to tell you how you should do it. What matters is that you build an XSD which can validate documents for further processing in the Indivo data pipeline.

Mapping the Schema to Data Models

In order to arrived at processed facts that can be queried and retrieved, you’ll need to a way to transform documents matching your schema into a form understood by Indivo. We call this tool (intuitively) a Transform, and you can learn how to build one here.

If the data in your new schema doesn’t fit into any of the Indivo Data Models, and you want to add a new data-model to Indivo, see Adding Custom Data-Models to Indivo.

Dropping the Schema into the Filesystem

Indivo schemas currently have the following layout on the filesystem:

```
indivo_server/
    indivo/
        ...
        schemas/
            utils/
            metadata/
            data/
                common/
                output/
                core/
                    sdmx/
                        schema.xsd
                        transform.[xslt | py]
                        sdmx.xml
                    ...
            contrib/
```

2.6. Indivo Schemas
The `indivo/schemas/data/core/` directory contains all of our built-in schemas, and you shouldn’t modify it. Since you are ‘contributing’ a schema to Indivo, add your schema to the `indivo/schemas/data/contrib/` directory. Simply:

- Create a new subdirectory under `indivo/schemas/data/contrib/`.

- Drop the following files into that directory:
  - `schema.xsd`: Your schema. This file **MUST BE NAMED SCHEMA.XSD** to be identified as a schema.
  - `transform.xslt` or `transform.py`: Your transform. This file **MUST BE NAMED ‘transform’** to be identified.
  - `sample.xml (optional)`: A sample document that should validate against your schema. This is optional, but is a good way to make sure your schema works as intended. If you have one or more sample xml files in your directory (you can name them anything, as long as the filename ends in `.xml’), you can make sure that they all validate by running:
    
    ```bash
cd indivo_server/indivo/schemas
python utils/validate.py
```

    
    This will validate all sample documents against their schema: you should see ‘ok’ at the end of each line of output if there were no errors.

- Restart Indivo for your changes to take effect. Your final directory structure should now look something like:

```
indivo_server/
  indivo/
  ...
  schemas/
    utils/
    metadata/
    data/
      common/
      output/
      core/
        sdmx/
          schema.xsd
          transform.[xslt | py]
          sdmx.xml
          ...
    contrib/
      your_schema/
        schema.xsd
        transform.[xslt | py]
        your_example1.xml
        your_example2.xml
        ...
```
3.1 Architecture Overview

This document provides a basic overview of the Indivo X system architecture. This document should be read before continuing on to Indivo Authentication and the Indivo API.

3.1.1 Basic Indivo Concepts

**Indivo Record**: the complete set of medical information stored by Indivo about a single individual.

**Indivo Account**: a username/password to log into Indivo. One account may be able to access any number of Indivo Records, and one Indivo Record may be accessible by multiple Indivo Accounts.

**Indivo Document**: a piece of medical information stored in an Indivo Record.

3.1.2 Components

Indivo X comprises multiple components, each running as its own web server. Small installations may choose to install multiple components on a single physical server. The Indivo X Server is the core of the system; other components, including the Indivo User Interface, can be easily substituted by custom implementations.
Indivo X Documentation, Release 2.0.0

**Indivo X Server**

For a given Indivo installation, the Indivo X server:

- stores all Indivo account information, as well as the medical records and documents,
- is responsible for authentication and authorization before granting access to Indivo data,
- exposes an API for access by administrative and user applications, and by the Indivo User Interface.

**Indivo User Interface / Indivo Chrome**

The Indivo User Interface, also known as the “Indivo Chrome”, implements the web-based visual interface that an Indivo user will view and use. The branding/colors/details of the user interface are all controlled by the Indivo Chrome. Indivo Chrome connects to Indivo X using the standard Indivo API, including some specific calls accessible only to the Chrome component.

Indivo X will ship with a default implementation of the Indivo Chrome which can be customized while maintaining a clean interface to the Indivo X API. Customizations are encouraged for re-branding or for entirely different devices, e.g. iPhone.

(The term “Chrome” is often used to describe the visual portions of a web browser that are part of the web browser itself, and not part of the web site content, e.g. the back button. Here, with “Indivo Chrome”, we mean the Indivo user interface that is part of the core Indivo service, not part of a user application that extends Indivo functionality.)
Administrative Application

An Admin Application can connect to Indivo X and

• create new Indivo accounts and records
• reset of passwords
• manage ownership of records, i.e. assigning an account as the owner of a record.

An admin application cannot access medical data, it can only manage a record’s metadata. An admin app is thus ideal for a hospital administrator, an Indivo help-desk staffer, a research administrator, etc.

User Application / Personal Health Application

A user application, or Personal Health Application, is an application that Indivo users manually add to their record to provide incremental functionality. Examples of PHA functionality include:

• Diabetes management
• Genomic data display
• Clinical trial matching and messaging

User applications generally provide a web interface to the Indivo user, while connecting to the user’s Indivo record directly with the Indivo X Server. Users are fully in control of what data a user application can access. They can, at any time, change those permissions or remove the application entirely. Thus, an Indivo user application connects to Indivo in much the same way that a Facebook application connects to Facebook.

3.1.3 Communication Protocols

All communication between components is over HTTPS, with an API that abides by the REST design philosophy. Authentication is via OAuth.

More about Indivo Authentication.

More about the Indivo API.

3.2 Indivo API

The Indivo API provides an interface for Personal Health Applications to extend the functionality of the Indivo PCHR. The Indivo API abides by the REST design pattern: it is stateless, re-uses existing HTTP constructs such as caching, compression and content negotiation, and generally uses URLs to represent hierarchical resources, e.g. documents within a medical record.

This document provides an introduction to the API, with in-depth explanations of related core Indivo concepts where applicable. For a full listing of all available Indivo API calls, see API Reference.

3.2.1 Overview

Personal Health Applications (PHAs) make HTTP calls to the Indivo API endpoint using the REST convention. OAuth is used to authenticate all calls, either in 2-legged mode for simple authentication, or in 3-legged mode when the PHA is making a call with delegated authentication, i.e. on behalf of the user.
Application Types

We consider three types of applications:

- **User Applications**, which individual Indivo users can add to their record.
- **Administrative Applications**, which are used to perform account and record manipulations.
- **UI Applications**, which provide the public user interface to Indivo features.

Per Indivo installation, there is a small handful of UI and administrative applications, and quite a number of user applications.

Terminology

A **record** is the single set of medical information that pertains to an individual. It is composed of **documents**, including a **demographics document** which details the individual’s contact information and name. A record can be accessed by one or more **accounts**.

**oAuth** is the authentication protocol used by Indivo. In 2-legged oAuth, the PHA (the oAuth **consumer**) makes calls to Indivo (the oAuth **service provider**) using a **consumer key** to identify itself, and a **consumer secret** to sign the request. In 3-legged oAuth, the PHA makes calls to Indivo to access medical information as delegated by the user, using an additional **token** and **token secret** that pertain to the specific Indivo record being accessed.

3.2.2 Authentication

All calls to Indivo are authenticated using **oAuth**.

We detail the authentication process at **Indivo Authentication**.

3.2.3 Design Patterns

Some common design patterns are repeated throughout the API. Not all of these patterns are necessarily 100% supported by all Indivo API implementations, though when they are they are consistent.

Email Addresses as Identifiers

Core accounts in Indivo X are identified by email addresses, because email addresses provide mechanisms for distributed identification and messaging. When an email address is included in a URL, it must be URL encoded, where the `@` sign turns into `%40`.

Paging/Filtering Results

When a list of results are returned, the URL ends in a `/` and the HTTP method is a **GET**, as is typical of REST design. In that case, Indivo X supports a generic query string that determines paging and ordering of the results:

```
?offset={offset}&limit={limit}&order_by={order_by}&status={document_status}&modified_since={modified_since}
```

- **offset** indicates which item number to start with, e.g. when getting a second batch of items.
- **limit** indicates the maximum number of items to return. This is used in combination with offset to accomplish paging.
- **order_by** is dependent on the fields returned in the list of items, and each call must thus define which fields are valid. Using an invalid field in order_by results in no effect on the output, as if order_by were absent.
• **status** can be used where applicable. It pertains to the status of documents and can currently be set to one of three options: ‘void’, ‘archived’ or ‘active’

• **modified_since** allows an application to look at items that have been modified since a given timestamp, so that incremental downloads may be possible.

**Querying Results**

As of the Beta3 release, calls that implement the basic paging operations above may also implement a more powerful **query interface**, also represented in the query string. In these cases (currently all of the minimal medical reports and the auditing calls), the following values may occur in the query string:

```
?offset={offset}&limit={limit}&order_by={order_by}&status={document_status}
```

These values function as before.

```
?group_by={group_field}&aggregate_by={aggregation_operator}*{aggregation_field}
```

**group_by** groups results by the specified field. It must be used in conjunction with **aggregate_by**, which aggregates the results by group, using the specified operation. If **aggregate_by** is passed without a **group_by** parameter, the aggregation is performed over the entire result set. Results that have been aggregated are returned in an aggregated format, not the typical reporting format.

```
?date_range={date_field}*{start_date}*{end_date}
```

date_range filters results and leaves only those with the specified field falling between **start_date** and **end_date**.

```
?date_group={date_field}*{time_increment}&?aggregate_by={aggregation_operator}*{aggregation_field}
```

date_group functions equivalently to **group_by**, except the groups are formed based on the values of the specified date field. For example, if the date field was ‘date_measured’, and the time increment was ‘month’, results would be returned grouped by the month of the date_measured field for each item. As with **group_by**, **date_group** must be used with an aggregator, and results are returned in an aggregated format.

```
?{FIELD}={VALUE}
```

This syntax adds additional filters to the query, returning only results having whose value for the property specified by ‘field’ matches ‘value’.

For each of these parameters, acceptable values for **{field}** are specified individually by the calls. A full listing of the minimal reporting fields, along with valid aggregation operators and date increments, may be found [here](#).

**External IDs**

When a resource is created, the Indivo API offers the ability to create this resource using a **PUT** with an **external_id** in the URL, so that the call is idempotent: if a failure occurs, the call can be repeated safely and only the resource will not be created on the second call if it was already created successfully during the first call.

An **external_id** is only valid within a particular PHA scope. Other PHAs cannot see the external_id of a given document if they didn’t create the document, and certainly cannot access the document by external_id.

Some API calls which involve both creating documents and retrieving them, such as:

```
PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/external/{APP_ID}/{EXTERNAL_ID}
```

For these calls, it can be confusing as to which document is referenced by an external id. In such cases, the following rule resolves confusion:

---

3.2. Indivo API
• The newly created document will always be assigned the passed `external_id`. The `external_id` will not be used to look up the existing document.

### 3.2.4 Managing Documents

Data stored in Indivo cannot by permanently deleted by default: the API enforces only appending data, not fully replacing it or removing it.

#### Reading Documents

**GET /records/{RECORD_ID}/documents/**, **GET /carenets/{CARENET_ID}/documents/**  
List documents within a record. Supports order by document metadata fields (see Indivo Document Metadata Schema).

The calls to **GET /records/{RECORD_ID}/documents/** and **GET /carenets/{CARENET_ID}/documents/** take a `type` querystring parameter, which filters the list of returned documents by their types.

A document’s `type` is (by default) the suffix of a URL that corresponds to the XML schema datatype, where the prefix is `http://indivo.org/vocab/xml/documents#`. Thus, type can be Medication, Lab, etc.

Indivo X supports storing XML documents whose datatype is not among the default Indivo X recommended types. In those cases, if the XML schema namespace doesn’t end in a `/` or `#`, then as is typical in the XML/RDF community, a `#` is used as delimiter in the URI. Examples of document types include:

- `http://indivo.org/xml/phr/medication#Medication` (Indivo 3.1 data type)

**GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}, GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID}**  
Fetch a single document.

**GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/meta, GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID}/meta**  
Fetch metadata about a single document, using its internal or external id.

**GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/versions/**  
List versions of a single document.

#### Writing Documents

**PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/label, PUT /records/{RECORD_ID}/documents/external/{APP_ID}/{EXTERNAL_ID}/label**  
Update a single document’s label.

**POST /records/{RECORD_ID}/documents/**, **PUT /records/{RECORD_ID}/documents/external/{APP_ID}/{EXTERNAL_ID}/**  
Create a new document, and possibly assign it an external id.

Medical data cannot be replaced wholesale, only versioned. Thus, this call will fail (with a 400 Bad Request error code) if a document already exists in the given record with the given external ID.

**POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace, PUT /records/{RECORD_ID}/documents/**  
Replace one document with a new document content. The existing document remains, but is marked suppressed and replaced by the new document.

Medical data cannot be replaced wholesale, only versioned. Thus, this call will fail (with a 400 Bad Request error code) if a document already exists in the given record with the given external ID.
Removing and Archiving Documents

Generally, documents in Indivo cannot be removed, they can only be versioned. However, mistakes happen, and Indivo must deal with these somehow. Also, information eventually is out of date or no longer relevant.

All such changes are encoded in the Indivo API as changes to document status.

**POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/set-status** Change the status of a document. The passed status defines what happens to the specified document:

- **void**: If a document is entered in error, it can be marked as voided to indicate that the data is invalid.
  
  Only active documents can be voided. Voided documents are still reachable, but their metadata indicates their status, and by default they are not listed in typical document listings.

- **archived**: If a document is no longer relevant, it can be archived so that it doesn’t show up by default. Archival is different from voiding in that an archived document is still considered medically correct, just not particularly relevant anymore.
  
  Archived documents are still reachable, but their metadata indicates their archival status, and by default they are not listed in typical document listings.

- **active**: An active document is readily usable and will appear in search listings by default. Setting a document to active status will unvoid a voided document, or unarchive an archived document.

**GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/status-history** A document can be voided, unvoided, archived, unarchived any number of times. The status change applies to the entire version lineage of a document. The history of statuses, in reverse chronological order, can be obtained using this call.

Relating Documents

It is often useful to relate documents, e.g. annotating a document, re-filling a prescription, connecting diagnoses to an encounter, etc. In Indivo X, these relations can be declared no matter the data type of the underlying document. An image of an X-ray might be related to an XML document that interprets it, but of course there is no room in the image file for a pointer. So all references are stored externally to the documents.

Relationship types are taken from a fixed list, including:

- interpretation
- annotation
- followup

Eventually, full URLs will be supported for relationship types. The fixed list of types will then correspond to http://indivo.org/vocab/documentrels#{rel_type}.

In the following calls, {DOCUMENT_ID} is the document being interpreted, and {OTHER_DOCUMENT_ID} or the POST content is the interpretation.

**PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/{OTHER_DOCUMENT_ID}** Create a new relationship of type REL_TYPE between the two passed documents.

**POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/, PUT /records/{RECORD_ID}**

Create a new document and immediately relate it to an existing document, possibly assigning an external id to the newly created document.

Medical data cannot be replaced wholesale, only versioned. Thus, this call will fail (with a 400 Bad Request error code) if a document already exists in the given record with the given external ID.

**GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/** List all documents related to the passed document by the relationship REL_TYPE.
**Special Documents**

Demographics documents are special in that there should only be one per record, and they should be easy to find.

See also:

*Indivo Document Demographics Schema*  The XML Schema for Indivo Demographics Data

GET /records/{RECORD_ID}/demographics, GET /carenets/{CARENET_ID}/demographics

Fetch demographics from a carenet or record. Depending on the requested response_format return data is formatted as SMART RDF/XML (default), SDMJ, or SDMX.

**3.2.5 Messaging and Notifications**

Indivo supports a lightweight notification framework as well as a heavier message inbox. For more information, see *Messaging and Notifications*.

**Messaging**

GET /accounts/{ACCOUNT_ID}/inbox/  List available messages. By default, only non-archived messages are returned.

GET /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}  Fetch a single message.

POST /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}/archive  Archive a message.

POST /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}/accept  Accept a message attachment. A user can accept an attachment from a message into their medical record. This creates a new document on their record containing the contents of the attachment.

POST /accounts/{ACCOUNT_ID}/inbox/  Send a message to an account.

POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}  Send a message to a record. Messages to records can have attached documents (specified by the num_attachments parameter) which then need to be uploaded separately. The message isn’t delivered until all of its attachments are uploaded.

Since Accounts, not Records, are the users who log into the system to view messages, there is no way to view messages in a record’s inbox. Rather, when a message is sent to a record, every account authorized to view the message is sent a copy of the message, which they can retrieve via their account inbox.

POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}  Upload an attachment to a message.

**Notifications**

Notifications are intended to be a lightweight system for applications to alert users of activity in the application. This is especially relevant for apps that use sharing functionality: an app might want to notify other users of the app about a given user’s activity in it. UI apps should display these notifications in a twitter-feed like interface (our reference UI call it the ‘healthfeed’).

POST /records/{RECORD_ID}/notifications/  Send a notification to a record. As with inbox messages, notifications are propagated to the accounts that are authorized to view the record.
GET /accounts/{ACCOUNT_EMAIL}/notifications/  List available notifications.

### 3.2.6 Application-Specific Storage

Application-specific storage is meant for bookkeeping by individual applications that is not specific to any given record. These documents can be deleted, since they are not part of any permanent medical record. All application-specific storage API calls behave like the prior document API calls, though the documents are accessible only to the application in question. Most implementations of the Indivo API will likely impose a quota on Applications to ensure they do not store large amounts of data in the application-specific storage. This quota may be application-specific, depending on what the application is approved to do.

Application-specific storage calls, since they don’t correspond to any given record, are all 2-legged oAuth calls.


GET /apps/{APP_ID}/documents/{DOCUMENT_ID}  Fetch a single application-specific document.

GET /apps/{APP_ID}/documents/{DOCUMENT_ID}/meta, GET /apps/{APP_ID}/documents/external/{EXTERNAL_ID}/meta  Fetch metadata about a single application-specific document, by its internal or external id.

POST /apps/{APP_ID}/documents/, PUT /apps/{APP_ID}/documents/external/{EXTERNAL_ID}  Create an application-specific document, possibly assigning it an external id.

As this is application-level storage, making this call with an external id will overwrite any existing document with the same external id.

PUT /apps/{APP_ID}/documents/{DOCUMENT_ID}/label  Update the label of an application-specific document.

DELETE /apps/{APP_ID}/documents/{DOCUMENT_ID}  Delete an application-specific document. Since these documents do not contain medical data, deleting them is acceptable.

### 3.2.7 Record-Application-Specific Storage

Record-application-specific storage is meant for bookkeeping by individual applications. These documents can be deleted, since they are not part of the permanent medical record. All record-application-specific storage API calls behave like the prior document API calls, though the documents are accessible only to the application in question. Most implementations of the Indivo API will likely impose a quota on Applications to ensure they do not store large amounts of data in the record-application-specific storage. This quota may be application-specific, depending on what the application is approved to do.

Record-Application-specific storage calls are all 3-legged oAuth calls.


GET /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID}  Fetch a single record-application-specific document.

GET /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID}/meta, GET /records/{RECORD_ID}/apps/{APP_ID}/documents/external/{EXTERNAL_ID}/meta  Fetch metadata about a single record-application-specific document, by its internal or external id.

POST /records/{RECORD_ID}/apps/{APP_ID}/documents/, PUT /records/{RECORD_ID}/apps/{APP_ID}/documents/external/{EXTERNAL_ID}  Create a record-application-specific document, possibly assigning it an external id.

As this is record-application-level storage, making this call with an external id will overwrite any existing document with the same external id.
PUT /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID}/label  Update the label of a record-application-specific document.

DELETE /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID}  Delete a record-application-specific document. Since these documents do not contain medical data, deleting them is acceptable.

3.2.8 Processed Medical Reports

Indivo processes documents into medical reports. Each report can be altered by the basic paging mechanism or the more complex query interface described above. Over time, new reports may be introduced. For now, we define these as the minimal set of reports. Fields supported by individual reports for the querying interface may be found here. Response formats correspond to the Indivo Reporting Schema, and individual reports fit their individual datatype’s schema (see Medical Documents). If a report is accessed via a carenet, only documents that are shared into the carenet will appear in the results.

GET /records/{RECORD_ID}/reports/minimal/equipment/, GET /carenets/{CARENET_ID}/reports/minimal/equipment/  List equipment for a given record.

GET /records/{RECORD_ID}/reports/minimal/labs/, GET /carenets/{CARENET_ID}/reports/minimal/labs/  List lab results for a given record.

GET /records/{RECORD_ID}/reports/minimal/measurements/{LAB_CODE}/, GET /carenets/{CARENET_ID}/reports/minimal/measurements/{LAB_CODE}/  List measurements for a given record.

GET /records/{RECORD_ID}/reports/minimal/procedures/, GET /carenets/{CARENET_ID}/reports/minimal/procedures/  List procedures for a given record.

GET /records/{RECORD_ID}/reports/minimal/simple-clinical-notes/, GET /carenets/{CARENET_ID}/reports/minimal/simple-clinical-notes/  List clinical notes for a given record.

3.2.9 SMART API Calls

As Indivo now supports the SMART API, the following calls are now available:

GET /records/{RECORD_ID}/{MODEL_NAME}/  Get a SMART RDF list of a patient’s medical data of type MODEL_NAME. Available data models are:

- allergies
- encounters
- fulfillments
- immunizations
- lab_results
- medications
- problems
- vital_signs

GET /apps/{APP_ID}/manifest, GET /apps/manifests/  Get SMART-style JSON manifests for one or all apps registered with this instance of Indivo.

GET /ontology  Get the ontology used by a SMART container

GET /capabilities/  Get the SMART capabilities for this instance of Indivo.

GET /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences  Get account preferences for a specific application.
PUT /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences  Set account preferences for a specific application. Overrides previous preferences.

DELETE /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences  Remove all account preferences for a specific application.

3.2.10 Generic Reports

Indivo provides the ability to run ‘generic’ reports over all data models. These reports support the API Query Interface, and provide an out of the box solution for reporting over core and contributed data models, with the possibility for customization.

GET /records/{RECORD_ID}/reports/{DATA_MODEL}/, GET /carenets/{CARENET_ID}/reports/{DATA_MODEL}/  List a patient’s medical data.

3.2.11 Coding Systems

A number of Indivo documents contain coded values. These can be based on UMLS, SNOMED, etc. Indivo provides a generic API for looking up coded values. This API is particularly built to support live autocomplete in JavaScript.

GET /codes/systems/  List available coding systems. Return data is in JSON format.

GET /codes/systems/{SHORT_NAME}/query  Search a coding system for a value.

3.2.12 Autonomous Apps API

Autonomous user applications are unlike standard user apps in that they may not have a user interface, and require access to records without an active user session. In order to authenticate against Indivo on behalf of records at any time, autonomous apps may make the following calls:

GET /apps/{APP_ID}/records/  Return a list of records which have enabled the app, and to which (therefore) the app can authenticate and acquire access.

POST /apps/{APP_ID}/records/{RECORD_ID}/access_token  Retrieve a valid access token providing the app with access to a record. This call will only succeed if the app is autonomous, and if the record has enabled the app.

Using this call, an autonomous app can retrieve a valid access token for any record on which it is enabled, without an active user session.

3.2.13 Administrative API

Admin applications have access to Indivo’s administrative API, which enables control and setup of records and accounts.

Account Administration

GET /accounts/{ACCOUNT_ID}  Get information about an account. The account_id must be in the form of an email address.

POST /accounts/{ACCOUNT_ID}/info-set  Set information about an account, including the name and contact email of the account holder.

GET /accounts/search  Search for accounts by name or contact email.
GET /accounts/{ACCOUNT_EMAIL}/records/ List available records on an account. Supports order by label.

POST /accounts/ Create an account.

The primary and secondary secret arguments are optional and are used for helping the user initialize their account securely. A primary secret is sent directly by Indivo X server to the user at their ACCOUNT_ID email address in the form of a URL with an embedded secret. A secondary secret is generated by Indivo X and made available to the admin application using the GET /accounts/{ACCOUNT_ID}/secret call for the account. If it is asked for in this call, it is required at account activation time right after the user clicks on the activation URL (aka the primary secret). A secondary secret makes sense only if a primary secret is also requested. That’s why it’s called “secondary.”

POST /accounts/{ACCOUNT_ID}/authsystems/ Add an authentication system to an account.

Accounts initially have no “authentication systems” attached to them. Over time, Indivo accounts will be usable with OpenID and other authentication systems. An account needs to enabled for each authentication system that we want to use for that account. The default system is “password”. Thus, this call, when used with the “password” system, will set up the password and username for a new user.

POST /accounts/{ACCOUNT_ID}/secret-resend Resend an account’s initialization URL (which contains the primary secret for the account). This is useful if the account holder loses the original email.

POST /accounts/{ACCOUNT_ID}/forgot-password Reset an account when its password is forgotten.

If a password is forgotten, the solution is to reset the account and email the user as with their initialization email. This will prevent logins until the new initialization URL is clicked, and the new password is entered.

This could be accomplished with separate calls to POST /accounts/{ACCOUNT_ID}/reset, which sets the account state to uninitialized and resets the account secrets, and POST /accounts/{ACCOUNT_ID}/secret-resend, but this call combines both actions.

Note that this call resets both the primary and secondary secrets. The user will need to be given this secondary secret in a channel other than email. If a User Interface Application performed this reset, then the secondary secret should display on screen while the primary secret is automatically sent by email. The user interface could obtain the secondary secret (which is short) by calling GET /accounts/{ACCOUNT_ID}/secret, but the call to POST /accounts/{ACCOUNT_ID}/forgot-password returns the secondary secret to avoid the extra call.

POST /accounts/{ACCOUNT_ID}/initialize/{PRIMARY_SECRET} Initialize a new account.

Initializing an account that has been reset requires both the primary and secondary secrets. The primary secret is sent in the URL, and the secondary secret should be collected by the user interface. Specifically, the recommended process is:

- Indivo Backend server sends the reinitialization URL to the user as:
  
  INDIVO_UI_APP_LOCATION/account/initialize/account_id/primary_secret

- An Indivo UI App checks that the requested account is indeed in uninitialized state and prompts the user for his secondary secret (which the user knows simply as the “secret”) and his desired username and password.

- The Indivo UI App initializes the account using this call.

- The Indivo UI app sets up the account with the built-in password authsystem using the username/password provided by the user and the API call POST /accounts/{ACCOUNT_ID}/authsystems/.

POST /accounts/{ACCOUNT_ID}/set-state Set an account’s state. Possible account states are:

- uninitialized: an account that has been created by an administrative application and has not been activated by the user yet (with their confirmation URL and code).

- active: a normal active account.
disabled: an account locked because of too many failed login attempts.

retired: an account that is no longer in use.

POST /accounts/{ACCOUNT_ID}/authsystems/password/set  Force an account’s password to a new value. This should be used only in the context of an account reinitialization.

POST /accounts/{ACCOUNT_ID}/authsystems/password/change  Allow a user to change an account password. The given old password must be correct for this change to succeed. This is a 3-legged call, since the user is the one driving this interaction (unlike POST /accounts/{ACCOUNT_ID}/authsystems/password/set, wherein the admin app is forcefully setting a password).

GET /accounts/{ACCOUNT_ID}/primary-secret  Fetch an account’s primary secret. This should be used very sparingly as the primary secret should rarely be seen outside of the Indivo backend.

Record Administration

GET /records/{RECORD_ID}  Get info about a single record.

GET /records/search  Search Indivo for existing records by record label.

GET /records/{RECORD_ID}/owner  Get the owner of a record

GET /records/{RECORD_ID}/apps/  List applications attached to a record. Supports order by name.

POST /records/, PUT /records/external/{APP_ID}/{EXTERNAL_ID}  Create a new record, possibly assigning it an external id. This call requires a valid Indivo Demographics Document in order to create the record.

PUT /records/{RECORD_ID}/owner  Set the owner of a record.

POST /records/{RECORD_ID}/apps/{APP_ID}/setup  Prime a record with a user app. This sets up an app to run against a record without user consent. It should be used only in cases where obtaining consent is impossible or unnecessary (i.e., at a hospital installation of Indivo, this call could be used to prime all new records with the syncer application that pulls data into Indivo from the hospital EMR).

PUT /records/{RECORD_ID}/apps/{APP_ID}  Enable an app to run against a record. This gives the app access to the entire record.

DELETE /records/{RECORD_ID}/apps/{APP_ID}  Remove a user app from a record.

3.2.14 Indivo Chrome / User Interface API

These API calls are reserved for the UI server, which is deeply trusted to authorized other applications, proxy the user’s credentials, etc. It’s only a separate server for modularity, otherwise it has the same level of trust as the backend Indivo server.

POST /oauth/internal/session_create  Create a web-session for a user. This call returns a session token that can be used to authenticate 3-legged calls on behalf of the user for the duration of a standard web session (30 minutes by default)

POST /oauth/internal/request_tokens/{REQUEST_TOKEN}/claim  Claim a request token on behalf of a user. Before a request token can be viewed at all, it has to be claimed by a user. This ensures that a request token can’t be partially used by one user and completed by another.

The session-based chrome authentication will indicate to the backend which Account to associate with this request token. Once this call has been made for a request token, a second call with different session authentication will fail. (A second call with the same user authentication will be just fine, we don’t want a reload to cause a problem.)
If the request token is bound to an Indivo record (because the PHA knew it was authorizing for a given record), and the claimant does not have administrative rights over the record, this call will fail and the request token will be invalidated.

GET /oauth/internal/request_tokens/{REQUEST_TOKEN}/info  Retrieve information about an OAuth request token.

When authorizing a request token, the Indivo UI needs to know what that token represents. Once the token is claimed, the request token yields information via this call.

This call can only be called with session authentication matching the Account which claimed the request token earlier.

If a record_id is present in the response data, then the kind element is also present and indicates:

- new: a new request for a PHA that has not been authorized for this record yet
- same: a request for a PHA that is already attached to the record and no new permissions are requested
- upgrade: a request for a PHA that is already attached to the record but that is asking for more permissions or more permissive usage of the data.

In the same case, the Chrome UI is allowed to immediately approve the request token. In other cases, the Chrome UI must explain to the user that new permissions or rights are being granted and prompt the user for approval.

POST /oauth/internal/request_tokens/{REQUEST_TOKEN}/approve  Approve a request token on behalf of a user.

If a user approves an app addition, then the Chrome UI server needs to let the backend know.

This call, if it succeeds with a 200 OK, will return the location to which the user’s browser should be redirected:

location={url_to_redirect_to}

This call’s session authentication must match that which claimed the request token. The record_id is the record to which the user is attaching the application (i.e. my child’s record, not my own.) If the request token was pre-bound to a record, this record_id parameter must match, or this will throw an error.

POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials  Get credentials for Connect-style Authentication for a user app, and authorize them on behalf of an account.

This call will return tokens that can be used to sign future API calls by the user app, proxied by the UI.

GET /accounts/{ACCOUNT_ID}/check-secrets/{PRIMARY_SECRET}  Check the primary and secondary secrets of an account.

GET /oauth/internal/surl-verify  Verify a signed URL.

In some cases, an Indivo app will sign a URL that directs the user to the Indivo UI. A prime example is the use of Indivo Chrome widgets, i.e. the Document Sharing widget, that apps can embed within their user interface to reuse functionality from Indivo Chrome. A signed URL looks like this:

/widgets/WidgetName?param1=foo&param2=bar&surl_timestamp={TIMESTAMP}&surl_token={TOKEN}&surl_sig={SIGNATURE}

The signature contained in surl_sig is effectively a signature on the rest of the URL. The signature algorithm is as follows:

1. An app, with OAuth access token TOKEN and OAuth access token secret SECRET, wishes to sign a URL.
2. The app generates the SURL secret that corresponds to this access token as follows:

\[ <SURL_SECRET> = \text{HMAC}(<TOKEN_SECRET>, "SURL-SECRET") \]
using base64 encoding, where the idea is to actually sign the string “SURL-SECRET” to obtain the SURL secret itself.

3. this SURL secret is then used to sign the URL, first by appending a timestamp, the SURL token, and then computing the signature:

\[ <\text{SURL\_SIG}> = \text{HMAC\{<\text{SURL\_SECRET}>, \\
"/widgets/WidgeName?...&surl\_timestamp=<\text{TIMESTAMP}>\&surl\_token=<\text{TOKEN}>" \} \] 
in base 64, then appending it as a query parameter surl_sig.

3.2.15 Sharing

Overview

We want to simplify sharing. Indivo has two main mechanisms for sharing patient records with other accounts: Full Shares and Carenets.

**Full Share** A user may choose to share the entirety of their record with another account. The recipient account will then have access to all data (past, present, and future) contained in the record, and will be able to run any apps that have been bound to the record. The recipient of a full share will also be able to add new applications to the record and run them against data in the record.

Similarly, a user may choose to add an application to their full record. This effectively creates a ‘full share’ of the record with that application: the app has access to all data in the record.

As an example, a teen user of Indivo might choose to set up a full share of his / her record with a parent or guardian.

**Carenets** Full shares are not very flexible: they are an all or nothing proposition. In cases where sharing data appropriately requires more granularity or complexity, Indivo provides *carenets*, which allow a record to specify groups of accounts and apps that all have transparent access to whatever data the record shares into the carenet.

By default, each record will have 3 simple carenets: physicians, family, and work/school.

As an example, a patient might create an ‘exercise’ carenet, into which they place:

- **data**: blood-pressure readings, pedometer output, and other data associated with maintaining a healthy lifestyle.
- **apps**: blood-pressure viewers, exercise-trackers, and other apps that help the patient organize and interact with their exercise data.
- **accounts**: The patient’s Primary Care Physician, personal trainer, friends, or any other person with an interest in helping the patient develop healthy exercise habits.

Now anyone on the *accounts* list can log into Indivo and run any app on the *apps* list against any data on the *data* list.

Data can be placed into carenets individually, or autoshares by Document type. Users can override the type-auto-sharing on a document-by-document basis.

**Autoshare** Documents of a certain type can be auto-shared, so that they are added to a carenet automatically when they are added to the record. When auto-share preferences are set for a type of document within a given carenet, these preferences apply to all documents that do not have an explicit sharing preference declared on them.

**Nevershare** A user should be able to ask that a specific document be “never shared”. This flag prevents any sharing, no matter what the auto-share rules may be.
Authorization into a CareNet

When an app is added, it is normally given, along with its oAuth token, an `xoauth_indivo_record_id` that the token corresponds to. If the app is added to a carenet instead of a record, the app will receive instead an `xoauth_indivo_carenet_id`.

Carenet-aware API calls

Many of the document and reporting calls that can be made on `/records/{RECORD_ID}` can be made on `/carenets/{CARENET_ID}`. Where applicable, such calls have been listed throughout this document.

Importantly, carenets are (at present) **READ-ONLY**. Accounts placed in carenets may view any data in the carenets, but we have not implemented any calls for them to modify or add to that data. In the future, carenets will be write-capable.

Sharing API

Full Shares

**GET** `/records/{RECORD_ID}/shares/` List accounts with which a record has created full shares. This call also lists user apps that have access to the full record, as such apps have the same access to data as an account with a full share.

**POST** `/records/{RECORD_ID}/shares/` Create a new full-record share with an account. The role_label is currently nothing more than that: a label. The label will come back in a call to GET `/records/{RECORD_ID}/shares/`.

**DELETE** `/records/{RECORD_ID}/shares/{ACCOUNT_ID}` Delete a full share.

Basic Carenet Calls

**GET** `/records/{RECORD_ID}/carenets/` List existing carenets on a record.

**GET** `/carenets/{CARENET_ID}/record` Fetch basic information about the record that a carenet belongs to.

**POST** `/records/{RECORD_ID}/carenets/` Create a new carenet on a record.

**POST** `/carenets/{CARENET_ID}/rename` Rename a carenet.

**DELETE** `/carenets/{CARENET_ID}` Delete a carenet. This will unshare all of the data in the carenet with all users and apps in the carenet.

Data in Carenets

Carenets are useless until data has been shared into them. Data can be shared explicitly at the granularity of individual documents, or implicitly at the granularity of document type.

**PUT** `/records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}` Place a document in a carenet.

When a document is explicitly shared with a carenet, it is no longer tied to the auto-sharing rules for that carenet. However, auto-sharing rules with other carenets still apply.
DELETE /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}
Remove a document from a carenet

When a document is explicitly UNshared from a carenet, it is no longer tied to the auto-sharing rules for that carenet. However, auto-sharing rules with other carenets still apply.

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/ List carenets where a document is present

The mode attribute indicates how this document is shared. explicit means that the sharing preferences for this document are explicitly set. bytype indicates that it was auto-shared by document type. Other modes may be enabled in the future.

The value attribute indicates a negative share with a carenet, meaning that the user explicitly wants this document not shared with this carenet, even if auto-share rules would otherwise share it. Obviously this only makes sense for explicit carenet-shares.

POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}/autoshare-revert
Revert a document to auto-share rules. This means that, for this carenet, this document reverts to automatic sharing rules. This might mean a removal of the share with this carenet, or an addition, or no effect. However, from this point on, the record-wide rules apply.

**Warning:** This call has not yet been implemented.

GET /records/{RECORD_ID}/autoshare/bytype/ List auto-sharing preferences for a given document type with a record. This call returns a list of carenets into which the document type is auto-shared.

GET /records/{RECORD_ID}/autoshare/bytype/all List all auto-sharing preferences for a record. This call returns a list of document types with the carenets into which each type is auto-shared.

POST /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/set Add an auto-share of a given document type into a given carenet. This share applies to all documents that do not have an explicit sharing preference declared on them.

POST /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/unset Remove an auto-share for a given document type from a given carenet.

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare Set the never share flag on a document. A user should be able to ask that a document be “never shared”. This flag prevents any sharing, no matter what the auto-share rules may be.

DELETE /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare Remove the never-share flag on a document.

**Apps in Carenets**

Users needs to be able to place apps inside carenets in addition to documents, so that other accounts can run the applications. There are no issues with read/write permissions here, as permissions are associated with the accounts in a carenet, not the apps.

GET /carenets/{CARENET_ID}/apps/ List all apps in a carenet.

PUT /carenets/{CARENET_ID}/apps/{PHA_EMAIL} Add an app to a carenet.

DELETE /carenets/{CARENET_ID}/apps/{PHA_EMAIL} Remove an app from a carenet.

GET /carenets/{CARENET_ID}/apps/{APP_ID}/permissions Check an app’s permissions within a carenet. Since permissions are currently handled on accounts, not apps, this call will always indicate that the app has full permissions on the carenet.

3.2. Indivo API
Accounts in Carenets

Users need to be able to place other accounts inside carenets so that they can share data and apps. When accounts are added to a carenet, they are assigned read/write permissions, which define what actions they can take on data in the carenet.

GET /carenets/{CARENET_ID}/accounts/  List all accounts in a carenet.

POST /carenets/{CARENET_ID}/accounts/  Add an account to a carenet.

DELETE /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID}  Remove an account from a carenet.

GET /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID}/permissions  Check an account’s permissions within a given carenet. This call will return a list of document types and whether the account may write to each one within the given carenet.

or now, the document type is always "*", since read/write permissioning is currently more granular than at the carenet level. We may ultimately permit permissioning by document types within a carenet, in which case this will all be more informative.

Building Carenet-aware Apps

Indivo apps are given a record_id and an access token that matches that record to read and write documents, read reports, annotate, etc. In a sharing scenario, apps must become carenet-aware.

Requirements

- An app should be easily placed within any number of carenets, i.e. physicians and family, but not work/school.
- When an app is activated on a given record, it must have access to no more data than the user who activated it. For example, if the owner selects the app, then the app may have access to the entire record. If the owner’s school nurse activates the app, the nurse should have access to only the data that is in the work/school carenet.
- There may be a need to further constrain an app, so that even if the owner activates the app, it should not be able to see every data type, or may be constrained to one of the carenets anyways. This is DEFERRED for now.
- We must not depend on app developers to properly partition information. If an app is active in both the Family and Physicians carenets, and knows that the record_id is the same in both cases, it may well intermix data without realizing it. This would be bad. We need to make it harder for apps to hurt the user.

Scenario

Alice owns her Indivo record and has shared it with Bob, her HR representative at work, placing Bob in the “Work/School” carenet. Alice is pregnant but does not wish to reveal this information to her co-workers just yet. She has added the “Pregnancy Tracker” app to her record, making it visible to her Family and Physician carenets, but not to her Work/School carenet. Alice has a history of depression, information which she has shared with her Physicians, but not with her Family.

Visible Apps

The “Pregnancy Tracker” app has been added to the Family and Physicians carenets, but not the Work/School carenet, so Bob cannot even see the application when he visits Alice’s record. This is enforced by the Indivo platform itself.
Activating and using an App

Charlie, Alice’s father, is eager to check up on his future grandchild’s progress. He logs into Indivo, selects Alice’s record. He sees “Pregnancy Tracker” because that app is visible to the Family carenet. He launches the app, and uses its functionality to track Alice’s progress, her fetus’s growth, her blood tests, etc. The process when launching the app is:

- Clicking on the app directs the IFRAME to the start_url for the pregnancy tracker. The app must receive an indication of which record is being accessed at this point. This cannot be the record_id alone, and we may not even want to include the record_id at all, otherwise the app might confuse this data with that accessible to Physicians later on. Thus, instead of passing record_id to the IFRAME, Indivo passes only carenet_id.
- The oAuth process begins with the carenet_id only as part of the request for a request token.
- Indivo checks that the logged-in-user has the right to access this carenet, and if so authorizes the token.
- The token is bound to that carenet only, and cannot be used on any other carenet.
- The app can make requests to
  /carenets/{carenet_id}/documents/...
  without using the record_id at all. It doesn’t need to know the record_id.
- When the app is later activated by a Physician, who does have access to Alice’s history of depression, the app gets a different carenet_id, and from that carenet has access to the documents including mental health.
- This is not fool-proof: we still probably need to give the app access to some record information that will yield a unique identifier using the name, DoB, etc... but at least the default behavior for the app will not allow error-prone tracking across carenets.

oAuth Mechanics

We start with:

- A CarenetAccount row that shares a record’s carenet with another account
- A Share row that indicates that an app has access to the record
- A CarenetPHA row that makes the app available in the carenet.

The oAuth process is then:

- PHA requests a request token with a carenet_id instead of a record_id as parameter.
- PHA needs to have a share into the record or into the specific carenet for this to succeed.
- The request token needs to keep track of the carenet, because the Share might be for the whole record.
- The user approving the request token should be in the carenet in question.
- The access token already stores the account of the person it’s proxying for, so that should be enough.

3.2.16 Auditing

As Indivo will be installed at HIPAA-compliant hospital sites, it is important that it be able to track system usage. The auditing system logs all incoming requests to Indivo that use the Indivo API. To learn more about auditing in Indivo, see the audit system’s documentation.

GET /records/{RECORD_ID}/audits/query/ Query the audit system. This call allows for general queries over the audit logs. The query is specified in the parameters via the API Query Interface, and returns results in the style of Medical Reports.
All subsequent calls are deprecated, but maintained (for now) for backwards compatibility.

GET /records/{RECORD_ID}/audits/ List all audit log entries where a given record was accessed.

Deprecated since version 1.0.0.

GET /records/{RECORD_ID}/audits/documents/{DOCUMENT_ID}/ List all audit log entries where a given document was accessed.

Deprecated since version 1.0.0.

GET /records/{RECORD_ID}/audits/documents/{DOCUMENT_ID}/functions/{FUNCTION_NAME}/ List all audit log entries where a given document was accessed via a given internal Django view function.

Deprecated since version 1.0.0.

### 3.3 API Reference

This page contains a full list of valid Indivo API calls, generated from the code. For a more detailed walkthrough of individual calls, see *Indivo API*.

In the access control for some calls, you might see the phrase *full control*. In Indivo, this is defined as follows:

**full control** A principal is said to be in full control of a record if either:

- The principal is the owner of the record
- There is a full share between the record and the principal.

**POST /accounts/**
Create a new account, and send out initialization emails.

**Short Name** account_create

**Accessible By** Any admin app.

**Formdata Parameters**

- primary_secret_p – 0 or 1: Does this account require a primary secret?
- secondary_secret_p – 0 or 1: Does this account require a secondary secret?
- contact_email – A valid email at which to reach the account holder.
- account_id – An identifier for the new account. Must be a valid email address. REQUIRED
- full_name – The full name to associate with the account.

**Returns** 200 OK with information about the new account on success, 400 Bad Request if ACCOUNT_ID isn’t passed or is already used.

Example Return Value:

```
<Account id="joeuser@indivo.example.org">
  <fullName>Joe User</fullName>
  <contactEmail>joeuser@gmail.com</contactEmail>
  <lastLoginAt>2010-05-04T15:34:23Z</lastLoginAt>
  <totalLoginCount>43</totalLoginCount>
  <failedLoginCount>0</failedLoginCount>
  <state>active</state>
  <lastStateChange>2009-04-03T13:12:12Z</lastStateChange>
```
GET /accounts/search
Search for accounts by name or email.

Short Name  account_search
Accessible By  Any admin app.
Query Parameters
- **fullname** – The full name of the account to search for
- **contact_email** – The contact email of the account to search for

Returns 200 OK with information about matching accounts, or 400 Bad Request if no search parameters are passed.

Example Return Value:

```xml
<Accounts>
  <Account id="joeuser@indivo.example.org">
    <fullName>Joe User</fullName>
    <contactEmail>joeuser@gmail.com</contactEmail>
    <lastLoginAt>2010-05-04T15:34:23Z</lastLoginAt>
    <totalLoginCount>43</totalLoginCount>
    <failedLoginCount>0</failedLoginCount>
    <state>active</state>
    <lastStateChange>2009-04-03T13:12:12Z</lastStateChange>
    
    <authSystem name="password" username="joeuser" />
    <authSystem name="hospital_sso" username="Joe_User" />
  </Account>
  ...
</Accounts>
```

GET /accounts/{ACCOUNT_EMAIL}
Display information about an account.

Short Name  account_info
Accessible By  Any admin app, or the Account owner.
URL Parameters
- **ACCOUNT_EMAIL** – The email identifier of the Indivo account

Returns 200 OK with information about the account

Example Return Value:

```xml
<Account id="joeuser@indivo.example.org">
  <fullName>Joe User</fullName>
  <contactEmail>joeuser@gmail.com</contactEmail>
  <lastLoginAt>2010-05-04T15:34:23Z</lastLoginAt>
  <totalLoginCount>43</totalLoginCount>
</Account>
```
POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials
Get oAuth credentials for an app to run in Connect or SMART REST mode.

**Short Name**  get_connect_credentials

**Accessible By**  The Account owner.

**URL Parameters**

- **ACCOUNT_EMAIL** – The email identifier of the Indivo account to authorize the connect credentials
- **PHA_EMAIL** – The email identifier of the Indivo user app to grant access via the connect credentials

**Formdata Parameters**

- **record_id** – The identifier of the Indivo Record to which to grant access via the connect credentials

**Returns**  200 OK with a set of credentials providing access for the app to the record, via Connect-Style Authentication and via Standard oAuth authentication. Additionally, the credentials include a precalculated oAuth Header that the app can use to access the record.

**Example Return Value:**

```xml
<Credentials>
  <App id="problems@apps.indivo.org" />
  <ConnectToken>1QzyGdx13Da</ConnectToken>
  <ConnectSecret>re3Qg4dxaf9</ConnectSecret>
  <APIBase>http://your_indivo_instance.org:8000</APIBase>
  <RESTToken>7qGer7dx4gC</RESTToken>
  <RESTSecret>5JpXb0G2g4u</RESTSecret>
  <OAuthHeader>OAuth realm="", oauth_version="1.0", oauth_consumer_key="problems%40apps.indivo.org", oauth_nonce="d2F66b9d99", oauth_timestamp="1334848404", oauth_signature_method="HMAC-SHA1", oauth_signature="... xA%3D"</OAuthHeader>
  <ExpiresAt>2012-07-04T00:00:00Z</ExpiresAt>
</Credentials>
```

New in version 2.0.0.

DELETE /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences
Delete all app-specific User Preferences for an account.

**Short Name**  delete_user_preferences

**Accessible By**  The Account owner.

**URL Parameters**

- **ACCOUNT_EMAIL** – The email identifier of the Indivo account
- **PHA_EMAIL** – The email identifier of the Indivo user app

**Returns**  200 Success
Example Return Value:

<ok/>

New in version 2.0.0.

---

**GET /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences**

Get app-specific User Preferences for an account.

**Short Name**  get_user_preferences

**Accessible By**  The Account owner.

**URL Parameters**

- **ACCOUNT_EMAIL** – The email identifier of the Indivo account
- **PHA_EMAIL** – The email identifier of the Indivo user app

**Returns**  200 OK, with app-specific user preferences

Example Return Value:

Preferences format is defined by the app setting the preferences, and will therefore vary.

New in version 2.0.0.

---

**PUT /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences**

Set app-specific User Preferences for an account.

**Short Name**  set_user_preferences

**Accessible By**  The Account owner.

**URL Parameters**

- **ACCOUNT_EMAIL** – The email identifier of the Indivo account
- **PHA_EMAIL** – The email identifier of the Indivo user app

**Returns**  200 Success

Example Return Value:

<ok/>

New in version 2.0.0.

---

**POST /accounts/{ACCOUNT_EMAIL}/authsystems/**

Add a new method of authentication to an account.

**Short Name**  account_authsystem_add

**Accessible By**  Any admin app.

**URL Parameters**

- **ACCOUNT_EMAIL** – The email identifier of the Indivo account

**Formdata Parameters**

- **username** – The username for this account
• **password** – The password for this account
  
• **system** – The identifier of the desired authsystem. `password` indicates the internal password system.

**Returns** 200 Success, 403 Forbidden if the indicated auth system doesn’t exist, and 400 Bad Request if a system and a username weren’t passed, or if the account is already registered with the passed system, or if the username is already taken for the passed authsystem.

Example Return Value:

```
<ok/>
```

---

**POST /accounts/{ACCOUNT_EMAIL}/authsystems/password/change**

Change a account’s password.

**Short Name** account_password_change

**Accessible By** The Account owner.

**URL Parameters**

• **ACCOUNT_EMAIL** – The email identifier of the Indivo account

**Formdata Parameters**

• **new** – The desired new password.

• **old** – The existing account password.

**Returns** 200 Success, 403 Forbidden if the old password didn’t validate, or 400 Bad Request if both a new and old password weren’t passed.

Example Return Value:

```
<ok/>
```

---

**POST /accounts/{ACCOUNT_EMAIL}/authsystems/password/set**

Force the password of an account to a given value.

**Short Name** account_password_set

**Accessible By** Any admin app.

**URL Parameters**

• **ACCOUNT_EMAIL** – The email identifier of the Indivo account

**Formdata Parameters**

• **password** – The new password to set.

**Returns** 200 Success, or 400 Bad Request if a new password wasn’t passed.

Example Return Value:

```
<ok/>
```

---

**POST /accounts/{ACCOUNT_EMAIL}/authsystems/password/set-username**

Force the username of an account to a given value.

**Short Name** account_username_set
Accessible By  Any admin app, or the Account owner.

URL Parameters

- ACCOUNT_EMAIL – The email identifier of the Indivo account

Formdata Parameters

- username – The new username to set.

Returns  200 Success, 400 Bad Request if a username wasn’t passed.

Example Return Value:

<ok/>

---

GET /accounts/{ACCOUNT_EMAIL}/check-secrets/{PRIMARY_SECRET}

Validate an account’s primary and secondary secrets.

Short Name  account_check_secrets

Accessible By  Any admin app.

URL Parameters

- ACCOUNT_EMAIL – The email identifier of the Indivo account
- PRIMARY_SECRET – A confirmation string sent securely to the patient from Indivo

Query Parameters

- secondary_secret – The secondary secret of the account to check.

Returns  200 Success, or 403 Forbidden if validation fails.

Example Return Value:

<ok/>

---

POST /accounts/{ACCOUNT_EMAIL}/forgot-password

Resets an account if the user has forgotten its password.

Short Name  account_forgot_password

Accessible By  Any admin app.

URL Parameters

- ACCOUNT_EMAIL – The email identifier of the Indivo account

Returns  :http:statuscode’200’ with the account’s new secondary secret, or 400 Bad Request if the account hasn’t yet been initialized.

Example Return Value:

<secret>123456</secret>

---

GET /accounts/{ACCOUNT_EMAIL}/inbox/

List messages in an account’s inbox.

Short Name  account_inbox

Accessible By  The Account owner.
URL Parameters
• ACCOUNT_EMAIL – The email identifier of the Indivo account

Query Parameters
• status – The account or document status to filter by
• order_by – See Query Operators
• limit – See Query Operators
• include_archive – 0 or 1: whether or not to include archived messages in the result set.
• offset – See Query Operators

Returns 200 OK, with a list of inbox messages.

Example Return Value:

```xml
<Message id="879">
  <sender>doctor@example.indivo.org</sender>
  <received_at>2010-09-04T14:12:12Z</received_at>
  <read_at>2010-09-04T17:13:24Z</read_at>
  <subject>your test results are looking good</subject>
  <severity>normal</severity>
  <record id="123" />
  <attachment num="1" type="http://indivo.org/vocab/xml/documents#Models" size="12546" />
</Message>
...
</Messages>

POST /accounts/{ACCOUNT_EMAIL}/inbox/
Send a message to an account.

Short Name account_send_message

Accessible By Any admin app.

URL Parameters
• ACCOUNT_EMAIL – The email identifier of the Indivo account

Formdata Parameters
• body – The message body. Defaults to [no body].
• severity – The importance of the message. Options are low, medium, high. Defaults to low.
• body_type – The formatting of the message body. Options are plaintext, markdown. Defaults to plaintext.
• message_id – An external identifier for the message, for idempotency.
• subject – The message subject. Defaults to [no subject].

Returns 200 Success with XML describing the message, or http:statuscode:400 if the passed message_id is a duplicate. Also emails account to alert them that a new message has arrived.

Example Return Value:
GET /accounts/{ACCOUNT_EMAIL}/inbox/{MESSAGE_ID}
Retrieve an individual message from an account’s inbox.

Short Name  account_inbox_message
Accessible By  The Account owner.

URL Parameters
- ACCOUNT_EMAIL – The email identifier of the Indivo account
- MESSAGE_ID – The unique identifier of the Indivo Message

Returns  200 OK, with XML describing the message.

Example Return Value:
<Message id="879">
  <sender>doctor@example.indivo.org</sender>
  <received_at>2010-09-04T14:12:12Z</received_at>
  <read_at>2010-09-04T17:13:24Z</read_at>
  <archived_at>2010-09-04T17:15:24Z</archived_at>
  <subject>your test results are looking good</subject>
  <body>Great results!
  It seems you’ll live forever!</body>
  <severity>normal</severity>
  <record id="123" />
  <attachment num="1" type="http://indivo.org/vocab/xml/documents#Models" size="12546" />
</Message>

POST /accounts/{ACCOUNT_EMAIL}/inbox/{MESSAGE_ID}/archive
Archive a message.

Short Name  account_message_archive
Accessible By  The Account owner.

URL Parameters
- ACCOUNT_EMAIL – The email identifier of the Indivo account
- MESSAGE_ID – The unique identifier of the Indivo Message

Returns  200 Success.

Example Return Value:
<ok/>
POST /accounts/{ACCOUNT_EMAIL}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}/accept

Accept a message attachment into the record it corresponds to.

Short Name account_inbox_message_attachment_accept

Accessible By The Account owner.

URL Parameters

• ATTACHMENT_NUM – The 1-indexed number corresponding to the message attachment
• ACCOUNT_EMAIL – The email identifier of the Indivo account
• MESSAGE_ID – The unique external identifier of the Indivo Message

Returns 200 Success, or 410 Gone if the attachment has already been saved.

Example Return Value:

<ok/>

POST /accounts/{ACCOUNT_EMAIL}/info-set

Set basic information about an account.

Short Name account_info_set

Accessible By Any admin app, or the Account owner.

URL Parameters

• ACCOUNT_EMAIL – The email identifier of the Indivo account

Formdata Parameters

• contact_email – A valid email at which to reach the account holder.
• full_name – The full name of the account.

Returns 200 OK, or 400 Bad Request if no parameters are passed in.

Example Return Value:

<ok/>

POST /accounts/{ACCOUNT_EMAIL}/initialize/{PRIMARY_SECRET}

Initialize an account, activating it.

Short Name account_initialize

Accessible By Any Indivo UI app.

URL Parameters

• ACCOUNT_EMAIL – The email identifier of the Indivo account
• PRIMARY_SECRET – A confirmation string sent securely to the patient from Indivo

Formdata Parameters

• secondary_secret –

Returns 200 Success, 403 Forbidden if the account has already been initialized or if secrets didn’t validate, and 400 Bad Request if a secondary secret was required but missing.

Example Return Value:
GET /accounts/{ACCOUNT_EMAIL}/notifications/
List an account’s notifications.

Short Name  account_notifications
Accessible By  The Account owner.

URL Parameters
  • ACCOUNT_EMAIL – The email identifier of the Indivo account
Query Parameters
  • status – The account or document status to filter by
  • limit – See Query Operators
  • order_by – See Query Operators
  • offset – See Query Operators

Returns 200 OK with a list of the account’s notifications.

Example Return Value:

<Notifications>
  <Notification id="468">
    <sender>labs@apps.indivo.org</sender>
    <received_at>2010-09-03T15:12:12Z</received_at>
    <content>A new lab result has been delivered to your account</content>
    <record id="123" label="Joe User" />
    <document id="579" label="Lab Test 2" />
  </Notification>
  ...
</Notifications>

GET /accounts/{ACCOUNT_EMAIL}/permissions/
List the carenets that an account has access to.

Short Name  account_permissions
Accessible By  The Account owner.

URL Parameters
  • ACCOUNT_EMAIL – The email identifier of the Indivo account

Returns 200 OK with a list of carenets.

Example Return Value:

<Carenets record_id="01234">
  <Carenet id="456" name="family" mode="explicit" />
  <Carenet id="567" name="school" mode="explicit" />
</Carenets>
GET /accounts/{ACCOUNT_EMAIL}/primary-secret
Display an account’s primary secret.

Short Name  account_primary_secret
Accessible By  Any admin app.

URL Parameters
•  ACCOUNT_EMAIL  – The email identifier of the Indivo account

Returns  200 OK, with the primary secret.

Example Return Value:

<secret>123absxzyasdgl3b</secret>

Deprecated since version 1.0.0: Avoid sending primary secrets over the wire. Instead, use GET
/accounts/{ACCOUNT_EMAIL}/check-secrets/{PRIMARY_SECRET}.

GET /accounts/{ACCOUNT_EMAIL}/records/
List all available records for an account.

Short Name  record_list
Accessible By  Any admin app, or the Account owner.

URL Parameters
•  ACCOUNT_EMAIL  – The email identifier of the Indivo account

Query Parameters
•  status  – The account or document status to filter by
•  order_by  – See Query Operators
•  limit  – See Query Operators
•  offset  – See Query Operators

Returns  200 OK, with a list of records owned or shared with the account.

Example Return Value:

<Records>
<Record id="123" label="John R. Smith" />
<Record id="234" label="John R. Smith Jr. (shared)" shared="true" role_label="Guardian" />
<Record id="345" label="Juanita R. Smith (carenet)" shared="true" carenet_id="678" carenet_name="family" />
...

</Records>

POST /accounts/{ACCOUNT_EMAIL}/reset
Reset an account to an uninitialized state.

Short Name  account_reset
Accessible By  Any admin app.

URL Parameters
•  ACCOUNT_EMAIL  – The email identifier of the Indivo account
Returns 200 Success.

Example Return Value:

<ok/>

---

GET /accounts/{ACCOUNT_EMAIL}/secret
Return the secondary secret of an account.

Short Name  account_secret
Accessible By  Any admin app.

URL Parameters
•  ACCOUNT_EMAIL – The email identifier of the Indivo account

Returns 200 OK, with the secondary secret.

Example Return Value:

<secret>123456</secret>

---

POST /accounts/{ACCOUNT_EMAIL}/secret-resend
Sends an account user their primary secret in case they lost it.

Short Name  account_resend_secret
Accessible By  Any admin app.

URL Parameters
•  ACCOUNT_EMAIL – The email identifier of the Indivo account

Returns 200 Success. Also emails the account with their new secret.

Example Return Value:

<ok/>

---

POST /accounts/{ACCOUNT_EMAIL}/set-state
Set the state of an account.

Short Name  account_set_state
Accessible By  Any admin app.

URL Parameters
•  ACCOUNT_EMAIL – The email identifier of the Indivo account

Formdata Parameters
•  state – The desired state of the account. Options are active, disabled, retired.

Returns 200 Success, or 403 Forbidden if the account has been retired and can no longer change state.

Example Return Value:

<ok/>
GET /apps/
List all available userapps.

  **Short Name** all_phas

  **Accessible By** Any principal in Indivo.

  **Returns** 200 OK, with a list of app manifests as JSON on success.

Example Return Value:

```json
[
  {
    "name" : "SMART Problems",
    "description" : "Display problems in a table view",
    "author" : "Josh Mandel, Children’s Hospital Boston",
    "id" : "problem-list@apps.smartplatforms.org",
    "version" : ".1a",
    "mode" : "ui",
    "scope" : "record",
    "requires" : {
      "http://smartplatforms.org/terms#Problem": {
        "methods": ["GET"]
      }
    }
  },
  ...
]
```

Changed in version 2.0.0: Apps are now returned as JSON manifests, not XML

---

GET /apps/manifests/
List SMART manifests for all available userapps.

  **Short Name** all_manifests

  **Accessible By** Any principal in Indivo.

  **Returns** SMART-style manifests for each app.

Example Return Value:

```json
[
  {
    "name" : "SMART Problems",
    "description" : "Display problems in a table view",
    "author" : "Josh Mandel, Children’s Hospital Boston",
    "id" : "problem-list@apps.smartplatforms.org",
    "version" : ".1a",
    "mode" : "ui",
```
"scope": "record",

"icon": "http://fda.gping.org:8012/framework/problem_list/icon.png",

"requires": {
  "http://smartplatforms.org/terms#Problem": {
    "methods": ["GET"]
  }
},

... other apps ...

} New in version 2.0.0.

DELETE /apps/{PHA_EMAIL}

Delete a userapp from Indivo.

**Short Name** pha_delete

**Accessible By** The user app itself.

**URL Parameters**

- **PHA_EMAIL** – The email identifier of the Indivo user app

**Returns** 200 Success.

Example Return Value:

<ok/>

GET /apps/{PHA_EMAIL}

Return a description of a single userapp.

**Short Name** pha

**Accessible By** Any principal in Indivo.

**URL Parameters**

- **PHA_EMAIL** – The email identifier of the Indivo user app

**Returns** 200 OK, with the app’s JSON manifest

Example Return Value:

{ "name": "SMART Problems",
  "description": "Display problems in a table view",
  "author": "Josh Mandel, Children's Hospital Boston",
  "id": "problem-list@apps.smartplatforms.org",
  "version": ".1a",

  "mode": "ui",
  "scope": "record",
}
GET /apps/{PHA_EMAIL}/documents/
List app-specific documents.

Short Name    app_document_list
Accessible By  A user app with an id matching the app email in the URL.

URL Parameters
• PHA_EMAIL – The email identifier of the Indivo user app

Query Parameters
• status – The account or document status to filter by
• order_by – See Query Operators
• type – The Indivo document type to filter by
• limit – See Query Operators
• offset – See Query Operators

Returns 200 OK with A list of documents, or http:statuscode:404 if an invalid type was passed in the querystring.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
    <label>HBA1C reading</label>
    <status>active</status>
    <nevershare>false</nevershare>
    <relatesTo>
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    </relatesTo>
  </latest>
</Document>
```

Changed in version 2.0.0: Apps are now returned as JSON manifests, not XML.
POST /apps/{PHA_EMAIL}/documents/
Create an app-specific Indivo document.

**Short Name** app_document_create

**Accessible By** A user app with an id matching the app email in the URL.

**URL Parameters**

- **PHA_EMAIL** – The email identifier of the Indivo user app

**Raw Data** The raw content of the document to create.

**Returns** 200 OK with the metadata of the created document, or 400 Bad Request if the new document failed validation.

**Example Return Value:**

```
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
    <createdAt>2009-05-04T17:05:33</createdAt>
    <creator id="steve@indivo.org" type="account">
        <fullname>Steve Zabak</fullname>
    </creator>
    <suppressedAt>2009-05-06T17:05:33</suppressedAt>
    <suppressor id="steve@indivo.org" type="account">
        <fullname>Steve Zabak</fullname>
    </suppressor>
    <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
    <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org"/>
    <label>HBA1C reading</label>
    <status>active</status>
    <nevershare>false</nevershare>
    <relatesTo>
        <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    </relatesTo>
    <isRelatedFrom>
        <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
    </isRelatedFrom>
</Document>
```

PUT /apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID}
Create an app-specific Indivo document with an associated external id.

**Short Name** app_document_create_or_update_ext

**Accessible By** A user app with an id matching the app email in the URL.

**URL Parameters**

- **EXTERNAL_ID** – The external identifier of the desired resource
- **PHA_EMAIL** – The email identifier of the Indivo user app
Raw Data  The raw content of the document to create.

Returns  200 OK with the metadata of the created or updated document, or 400 Bad Request if the passed content didn’t validate.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-BbBe-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d54d6" creationTime="2009-05-04T17:05:33" creationBy="steve@indivo.org" type="account">
  <creator id="problems@apps.indivo.org" type="pha">
    <fullname>Steve Zabak</fullname>
  </creator>
  <latest id="14c81023-c84f-496d-BbBe-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" type="account">
    <label>HBA1C reading</label>
    <status>active</status>
    <neverShare>false</neverShare>
  </latest>
</Document>
```

GET /apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID}/meta

Fetch the metadata of an app-specific document identified by external id.

Short Name  app_document_meta_ext

Accessible By  A user app with an id matching the app email in the URL.

URL Parameters

- EXTERNAL_ID  – The external identifier of the desired resource
- PHA_EMAIL  – The email identifier of the Indivo user app

Returns 200 OK with metadata describing the specified document, or http:statuscode:404 if the external_id is invalid.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-BbBe-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d54d6" creationTime="2009-05-04T17:05:33" creationBy="steve@indivo.org" type="account">
  <creator id="problems@apps.indivo.org" type="pha">
    <fullname>Steve Zabak</fullname>
  </creator>
  <latest id="14c81023-c84f-496d-BbBe-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" type="account">
    <label>HBA1C reading</label>
    <status>active</status>
    <neverShare>false</neverShare>
  </latest>
</Document>
```

DELETE /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}

Delete an app-specific document.
**Short Name**  app_document_delete

**Accessible By**  A user app with an id matching the app email in the URL.

**URL Parameters**

- **PHA_EMAIL**  – The email identifier of the Indivo user app
- **DOCUMENT_ID**  – The unique identifier of the Indivo document

**Returns**  200 Success, or 404 Not Found if DOCUMENT_ID is invalid.

Example Return Value:

```xml
<ok>
</ok>
```

**GET /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}**

Retrive an app-specific document.

**Short Name**  app_specific_document

**Accessible By**  A user app with an id matching the app email in the URL.

**URL Parameters**

- **PHA_EMAIL**  – The email identifier of the Indivo user app
- **DOCUMENT_ID**  – The unique identifier of the Indivo document

**Returns**  200 OK with the raw content of the document, or 404 Not Found if the document could not be found.

Example Return Value:

```xml
<DefaultProblemsPreferences record_id="123">
  <Preference name="hide_void" value="true" />
  <Preference name="show_rels" value="false" />
</DefaultProblemsPreferences>
```

**PUT /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}**

Create or Overwrite an app-specific Indivo document.

**Short Name**  app_document_create_or_update

**Accessible By**  A user app with an id matching the app email in the URL.

**URL Parameters**

- **PHA_EMAIL**  – The email identifier of the Indivo user app
- **DOCUMENT_ID**  – The unique identifier of the Indivo document

**Raw Data**  The raw content of the document to create.

**Returns**  200 OK with metadata describing the created or updated document, or 400 Bad Request if the passed content didn’t validate.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="problems@apps.indivo.org" type="pha">
  </creator>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
</Document>
```
PUT /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/label
Set the label of an app-specific document.

**Short Name**  app_document_label

**Accessible By**  A user app with an id matching the app email in the URL.

**URL Parameters**
- **PHA_EMAIL** – The email identifier of the Indivo user app
- **DOCUMENT_ID** – The unique identifier of the Indivo document

**Raw Data**  The new label for the document

**Returns** 200 OK with metadata describing the re-labeled document, or 404 Not Found if DOCUMENT_ID is invalid.

Example Return Value:

```
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org"/>
  <label>RELABELED: New HBA1C reading</label>
  <status>active</status>
  <nevershare>false</nevershare>
  <relatesTo>
    <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
  </relatesTo>
  <isRelatedFrom>
    <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
  </isRelatedFrom>
</Document>
```

GET /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/meta
Fetch the metadata of an app-specific document.

**Short Name**  app_document_meta

**Accessible By**  A user app with an id matching the app email in the URL.

**URL Parameters**
- **PHA_EMAIL** – The email identifier of the Indivo user app
• **DOCUMENT_ID** – The unique identifier of the Indivo document

**Returns** 200 OK with the document metadata, or 404 Not Found if DOCUMENT_ID is invalid.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d547b1" createdAt="2009-05-04T17:05:33" createdBy="steve@indivo.org" type="account">
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" type="account">
    <label>HBA1C reading</label>
    <status>active</status>
    <nevershare>false</nevershare>
    <relatesTo>
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
      <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
    </relatesTo>
    <isRelatedFrom>
      <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
    </isRelatedFrom>
  </latest>
</Document>
```

**GET /apps/{PHA_EMAIL}/manifest**

Return a SMART manifest for a single userapp.

**Short Name** app_manifest

**Accessible By** Any principal in Indivo.

**URL Parameters**

• **PHA_EMAIL** – The email identifier of the Indivo user app

**Returns** A SMART-style manifest for the app.

Example Return Value:

```json
{
  "name": "SMART Problems",
  "description": "Display problems in a table view",
  "author": "Josh Mandel, Children’s Hospital Boston",
  "id": "problem-list@apps.smartplatforms.org",
  "version": "0.1a",
  "mode": "ui",
  "scope": "record",
  "icon": "http://fda.gping.org:8012/framework/problem_list/icon.png",
  "requires": {
    "http://smartplatforms.org/terms#Problem": {
      "methods": ["GET"]
  }
```
GET /apps/{PHA_EMAIL}/records/
Return a list of all records that have this pha enabled.

Short Name  app_record_list
Accessible By  Any autonomous user app.

URL Parameters

- PHA_EMAIL  – The email identifier of the Indivo user app

Returns  200 OK with a list of records on success.

Example Return Value:

```xml
<Records>
  <Record id='123' label='John R. Smith' />
  <Record id='234' label='Frank Frankson' />
...
</Records>
```

New in version 1.0.0.

POST /apps/{PHA_EMAIL}/records/{RECORD_ID}/access_token
Fetch an access token for an autonomous app to access a record.

Short Name  autonomous_access_token
Accessible By  An autonomous user app with a record on which the app is authorized to run.

URL Parameters

- RECORD_ID  – The id string associated with the Indivo record
- PHA_EMAIL  – The email identifier of the Indivo user app

Returns  200 OK with a valid access token for the app bound to the record on success.

Example Return Value:

```
oauth_token=abcd1fw3gasdg3oaouth_token_secret=jgr1hre4291hfjasxoaouth_indivo_record_id=123
```

New in version 1.0.0.

GET /capabilities/
SMART Capabilities

Short Name  smart_capabilities
Accessible By  Any principal in Indivo.

Returns  JSON formatted SMART capabilities
Example Return Value:

```
{
    "http://smartplatforms.org/terms#Demographics": {
        "methods": [
            "GET"
        ]
    },
    "http://smartplatforms.org/terms#Encounter": {
        "methods": [
            "GET"
        ]
    },
    "http://smartplatforms.org/terms#VitalSigns": {
        "methods": [
            "GET"
        ]
    }
}
```

New in version 2.0.0.

---

**DELETE /carenets/{CARENET_ID}**

Delete a carenet.

**Short Name**  carenet_delete

**Accessible By**  A principal in full control of the carenet’s record.

**URL Parameters**

- **CARENET_ID** – The id string associated with the Indivo carenet

**Returns**  200 Success.

Example Return Value:

```
<ok/>
```

---

**GET /carenets/{CARENET_ID}/accounts/**

List the accounts in a carenet.

**Short Name**  carenet_account_list

**Accessible By**  A principal in the carenet, in full control of the carenet’s record, or any admin app.

**URL Parameters**

- **CARENET_ID** – The id string associated with the Indivo carenet

**Returns**  200 OK with a list of accounts in the specified carenet.

Example Return Value:

```
<CarenetAccounts>
    <CarenetAccount id="johndoe@indivo.org" fullName="John Doe" write="true" />
    ...
</CarenetAccounts>
```
POST /carenets/{CARENET_ID}/accounts/
Add an account to a carenet.

Short Name  carenet_account_create
Accessible By  A principal in full control of the carenet’s record.

URL Parameters
  • CARENET_ID – The id string associated with the Indivo carenet

Formdata Parameters
  • write – true or false. Whether this account can write to the carenet.
  • account_id – An identifier for the account. Must be a valid email address.

Returns  200 Success, 404 Not Found if the specified account or carenet don’t exist, or 400 Bad Request if an account_id isn’t passed.

Example Return Value:
<ok/>

DELETE /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID}
Remove an account from a carenet.

Short Name  carenet_account_delete
Accessible By  A principal in full control of the carenet’s record.

URL Parameters
  • ACCOUNT_ID – The email identifier of the Indivo account
  • CARENET_ID – The id string associated with the Indivo carenet

Returns  200 Success, or 404 Not Found if either the passed account or the passed carenet doesn’t exist.

Example Return Value:
<ok/>

GET /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID}/permissions
List the permissions of an account within a carenet.

Short Name  carenet_account_permissions
Accessible By  A user app with access to the carenet and proxying the account, a principal in full control of the carenet’s record, or any admin app.

URL Parameters
  • ACCOUNT_ID – The email identifier of the Indivo account
  • CARENET_ID – The id string associated with the Indivo carenet

Returns  200 OK with a list of document types that the account can access within a carenet. Currently always returns all document types.

Example Return Value:
GET /carenets/{CARENET_ID}/apps/
List Apps within a given carenet.

**Short Name** carenet_apps_list

**Accessible By** A principal in the carenet, in full control of the carenet’s record, or any admin app.

**URL Parameters**
- CARENET_ID – The id string associated with the Indivo carenet

**Returns** 200 OK with manifests for the apps on success.

Example Return Value:

```json
[
  {
    "name": "SMART Problems",
    "description": "Display problems in a table view",
    "author": "Josh Mandel, Children’s Hospital Boston",
    "id": "problem-list@apps.smartplatforms.org",
    "version": ".1a",
    "mode": "ui",
    "scope": "record",
    "icon": "http://fda.gping.org:8012/framework/problem_list/icon.png",
    "requires": {
      "http://smartplatforms.org/terms#Problem": {
        "methods": ["GET"]
      }
    }
  },
  ...
]
```

Changed in version 2.0.0: Apps are now returned as JSON manifests, not XML

DELETE /carenets/{CARENET_ID}/apps/{PHA_EMAIL}
Remove an app from a given carenet.

**Short Name** carenet_apps_delete

**Accessible By** A principal in full control of the carenet’s record.

**URL Parameters**
- PHA_EMAIL – The email identifier of the Indivo user app
- CARENET_ID – The id string associated with the Indivo carenet

**Returns** 200 Success.
Example Return Value:

<ok/>

**PUT /carenets/{CARENET_ID}/apps/{PHA_EMAIL}**

Add an app to a carenet

**Short Name** carenet_apps_create  
**Accessible By** A principal in full control of the carenet’s record.

**URL Parameters**

- **PHA_EMAIL** – The email identifier of the Indivo user app  
- **CARENET_ID** – The id string associated with the Indivo carenet

**Returns** 200 Success, or 400 Bad Request if the passed PHA is autonomous (autonomous apps can’t be scoped to carenets).

Example Return Value:

<ok/>

**GET /carenets/{CARENET_ID}/apps/{PHA_EMAIL}/permissions**

Retrieve the permissions for an app within a carenet. NOT IMPLEMENTED.

**Short Name** carenet_app_permissions  
**Accessible By** Nobody

**URL Parameters**

- **PHA_EMAIL** – The email identifier of the Indivo user app  
- **CARENET_ID** – The id string associated with the Indivo carenet

**Returns** 200 OK. This call is unimplemented, and has no effect.

Example Return Value:

<ok/>

**Todo**

The API Call ‘GET /carenets/{0}/apps/{1}/permissions’ is not yet implemented.

**GET /carenets/{CARENET_ID}/demographics**

Read demographics from a carenet.

**Short Name** read_demographics_carenet  
**Accessible By** A user app with access to the carenet or the entire carenet’s record, an account in the carenet or in control of the record, or any admin app.

**URL Parameters**

- **CARENET_ID** – The id string associated with the Indivo carenet

**Query Parameters**
• **response_format** – one of `application/rdf+xml` (SMART RDF/XML), `application/json` (SDMJ), or `application/xml` (SDMX). Default is `application/rdf+xml`

**Returns** 200 OK with demographics formatted in the requested response_format (default `application/rdf+xml`) on success, 404 Not Found when no demographics found, and 400 Bad Request if response_format is invalid

Example Return Value:

```
application/rdf+xml:

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:dcterms="http://purl.org/dc/terms/"
         xmlns:foaf="http://xmlns.com/foaf/0.1/
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:sp="http://smartplatforms.org/terms#"
         xmlns:v="http://www.w3.org/2006/vcard/ns#">
  <rdf:Description rdf:nodeID="_6730841b-05df-445f-8695-ed64197f4e6a">
    <v:family-name>William</v:family-name>
    <v:given-name>Robinson</v:given-name>
  </rdf:Description>
  <rdf:Description rdf:nodeID="_bcf66b59-e438-49b2-b572-99af4319b297">
    <rdf:value>800-870-3011</rdf:value>
    <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Tel"/>
    <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Home"/>
    <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Pref"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="_8cbe3da1-fb53-4d31-80b6-19e0d04220ad">
    <dcterms:identifier>http://indivo.org/records/96ff9eb2-3b18-4a0e-9df8-5a731b96d5d6</dcterms:identifier>
    <sp:system>Indivo Record</sp:system>
    <dcterms:title>Indivo Record 96ff9eb2-3b18-4a0e-9df8-5a731b96d5d6</dcterms:title>
    <rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="_451ade87-b519-4c92-8d07-2bbfcb23999c">
    <v:tel rdf:nodeID="_bcf66b59-e438-49b2-b572-99af4319b297"/>
    <v:tel rdf:nodeID="_451ade87-b519-4c92-8d07-2bbfcb23999c"/>
    <sp:email>william.robinson@example.com</sp:email>
    <sp:gender>male</sp:gender>
    <sp:preferredLanguage>EN</sp:preferredLanguage>
    <v:n rdf:nodeID="_6730841b-05df-445f-8695-ed64197f4e6a"/>
    <sp:medicalRecordNumber rdf:nodeID="_8cbe3da1-fb53-4d31-80b6-19e0d04220ad"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="_9f06ee63-3704-4b2a-9c2a-109cc9c99a57">
    <v:street-address>23 Church Rd</v:street-address>
    <v:country>USA</v:country>
  </rdf:Description>
</rdf:RDF>
```
application/xml:

<Models>
  <Model name="Demographics" documentId="44190967-cbaa-43a7-a98c-9f97f094ef2b">
    <Field name="bday">1965-08-09</Field>
    <Field name="email">william.robinson@example.com</Field>
    <Field name="gender">male</Field>
    <Field name="preferred_language">EN</Field>
    <Field name="race"/>
    <Field name="name_given">Robinson</Field>
    <Field name="name_suffix"/>
    <Field name="name_family">William</Field>
    <Field name="name_prefix"/>
    <Field name="tel_2_type">h</Field>
    <Field name="tel_2_preferred_p">true</Field>
    <Field name="tel_2_number">800-870-3011</Field>
    <Field name="adr_region">OK</Field>
    <Field name="adr_country">USA</Field>
    <Field name="adr_postalcode">74008</Field>
    <Field name="adr_city">Bixby</Field>
    <Field name="adr_street">23 Church Rd</Field>
    <Field name="tel_1_type">h</Field>
    <Field name="tel_1_preferred_p">true</Field>
    <Field name="tel_1_number">800-870-3011</Field>
  </Model>
</Models>

application/json:

[ {
  "__modelname__": "Demographics",
  "__documentid__": "44190967-cbaa-43a7-a98c-9f97f094ef2b",
  "name_given": "Robinson",
  "name_family": "William",
  "name_prefix": null,
  "name_suffix": null,
  "gender": "male",
  "race": null,
  "ethnicity": null,
  "bday": "1965-08-09",
  "email": "william.robinson@example.com",
  "preferred_language": "EN",
  "tel_1_type": "h",
  "tel_1_number": "800-870-3011",
  "tel_1_preferred_p": true,
  "tel_2_type": "h",
  "tel_2_number": "800-870-3011",
  "tel_2_preferred_p": true,
  "adr_street": "23 Church Rd",
  "adr_city": "Bixby",
} ]
New in version 2.0.0.

GET /carenets/{CARENET_ID}/documents/
List documents from a given carenet.

**Short Name** carenet_document_list

**Accessible By** A user app with access to the carenet or the entire carenet’s record, or an account in the carenet or in control of the record.

**URL Parameters**
- **CARENET_ID** – The id string associated with the Indivo carenet

**Query Parameters**
- **type** – The Indivo document type to filter by

**Returns** 200 OK with a document list on success, 404 Not Found if type doesn’t exist.

Example Return Value:

```xml
<Documents record_id="123" total_document_count="3" pha=""
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d4" createdAt="2009-05-04T17:05:33" createdBy="steve@indivo.org" type="account"
  <fullname>Steve Zabak</fullname>
</creator>
<suppressedAt>2009-05-06T17:05:33</suppressedAt>
<suppressor id="steve@indivo.org" type="account"
  <fullname>Steve Zabak</fullname>
</suppressor>
<original id="14c81023-c84f-496d-8b8e-9438280441d3" />
<latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org"
  <label>HBA1C reading</label>
  <status>active</status>
  <nevershare>false</nevershare>
  <relatesTo>
    <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
  </relatesTo>
  <isRelatedFrom>
    <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
  </isRelatedFrom>
</Document>
...
</Documents>
```
Short Name  carenet_document

Accessible By  A user app with access to the carenet or the entire carenet’s record, or an account in the carenet or in control of the record.

URL Parameters

• DOCUMENT_ID – The unique identifier of the Indivo document
• CARENET_ID – The id string associated with the Indivo carenet

Returns 200 OK with the document content on success, 404 Not Found if document_id is invalid or if the document is not shared in the carenet.

Example Return Value:

<ExampleDocument>
  <content>That’s my content</content>
  <otherField attr="val" />
</ExampleDocument>

GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID}/meta

Fetch the metadata of a record-specific document via a carenet.

Short Name  carenet_document_meta

Accessible By  A user app with access to the carenet or the entire carenet’s record, or an account in the carenet or in control of the record.

URL Parameters

• DOCUMENT_ID – The unique identifier of the Indivo document
• CARENET_ID – The id string associated with the Indivo carenet

Returns 200 OK with the document’s metadata, or 404 Not Found if document_id doesn’t identify an existing document in the carenet.

Example Return Value:

<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" label="HBA1C reading" status="active" nevershare="false">
    <relatesTo>
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
      <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
    </relatesTo>
    <isRelatedFrom>
      <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
    </isRelatedFrom>
  </latest>
</Document>
GET /carenets/{CARENET_ID}/record
Get basic information about the record to which a carenet belongs.

**Short Name** carenet_record

**Accessible By** A principal in the carenet, in full control of the carenet’s record, or any admin app.

**URL Parameters**
- CARENET_ID – The id string associated with the Indivo carenet

**Returns** 200 OK with XML describing the record.

Example Return Value:

```xml
<Record id="123" label="Joe User">
  <demographics document_id="467" />
  <created at="2010-10-23T10:23:34Z" by="indivoconnector@apps.indivo.org" />
</Record>
```

POST /carenets/{CARENET_ID}/rename
Change a carenet’s name.

**Short Name** carenet_rename

**Accessible By** A principal in full control of the carenet’s record.

**URL Parameters**
- CARENET_ID – The id string associated with the Indivo carenet

**Formdata Parameters**
- name – The new name for the carenet.

**Returns** 200 OK with XML describing the renamed carenet on success, 400 Bad Request if name wasn’t passed or if a carenet named name already exists on this record.

Example Return Value:

```xml
<Carenets record_id="123">
  <Carenet id="789" name="Work/School" mode="explicit" />
</Carenets>
```

GET /carenets/{CARENET_ID}/reports/minimal/equipment/
List the equipment data for a given carenet.

**Short Name** carenet_equipment_list

**Accessible By** A user app with access to the carenet or the entire carenet’s record, or an account in the carenet or in control of the record.

**URL Parameters**
- CARENET_ID – The id string associated with the Indivo carenet

**Query Parameters**
- status – The account or document status to filter by
- {FIELD} – See Query Operators, Valid Query Fields
• date_group – See Query Operators
• group_by – See Query Operators
• order_by – See Query Operators
• aggregate_by – See Query Operators
• date_range – See Query Operators
• limit – See Query Operators
• offset – See Query Operators

Returns 200 OK with a list of equipment, or 400 Bad Request if any invalid query parameters were passed.

Example Return Value:

```xml
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*" />
    <Filters>
      <Filter name="equipment_name" value="pacemaker"/>
    </Filters>
  </QueryParams>
  <Report>
    <Item>
      <Equipment xmlns="http://indivo.org/vocab/xml/documents#">
        <dateStarted>2009-02-05</dateStarted>
        <dateStopped>2010-06-12</dateStopped>
        <type>cardiac</type>
        <name>Pacemaker</name>
        <vendor>Acme Medical Devices</vendor>
        <id>167-ABC-23</id>
        <description>it works</description>
        <specification>blah blah blah</specification>
      </Equipment>
    </Item>
  </Report>
  ...
</Reports>
```

GET /carenets/{CARENET_ID}/reports/minimal/measurements/{LAB_CODE}/

List the measurement data for a given carenet.
Short Name  carenet_measurement_list

Accessible By  A user app with access to the carenet or the entire carenet’s record, or an account in the carenet or in control of the record.

URL Parameters
- CARENET_ID – The id string associated with the Indivo carenet
- LAB_CODE – The identifier corresponding to the measurement being made.

Query Parameters
- status – The account or document status to filter by
- [FIELD] – See Query Operators, Valid Query Fields
- date_group – See Query Operators
- group_by – See Query Operators
- order_by – See Query Operators
- aggregate_by – See Query Operators
- date_range – See Query Operators
- limit – See Query Operators
- offset – See Query Operators

Returns  200 OK with a list of measurements, or 400 Bad Request if any invalid query parameters were passed.

Example Return Value:

```xml
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*" />
    <Filters>
      <Filter name="lab_type" value="hematology"/>
    </Filters>
  </QueryParams>
  <Report>
    <Meta>
      <Document id="261ca370-927f-41af-b001-7b615c7a468e" type="http://indivo.org/vocab/xml/documents#Measurement" size="1653" digest="0799971784e5a2d199cd6585415a8cd57f7bf9e4f8c8f74ef67a1009a1481cd6" record_id="">
        <createdAt>2011-05-02T17:48:13Z</createdAt>
        <creator id="mymail@mail.ma" type="Account">
          <fullname>full name</fullname>
        </creator>
        <original id="261ca370-927f-41af-b001-7b615c7a468e"/>
        <label>testing</label>
        <status>active</status>
        <nevershare>false</nevershare>
      </Document>
    </Meta>
    <Item>
      <Measurement id="1234" value="120" type="blood pressure systolic" datetime="2011-03-02T00:00:00Z" unit="mmHg" source_doc="3456"/>
    </Item>
  </Report>
  ...
</Reports>
```
GET /carenets/{CARENET_ID}/reports/minimal/procedures/
List the procedure data for a given carenet.

**Short Name** carenet_procedure_list

**Accessible By** A user app with access to the carenet or the entire carenet’s record, or an account in
the carenet or in control of the record.

**URL Parameters**
- CARENET_ID – The id string associated with the Indivo carenet

**Query Parameters**
- status – The account or document status to filter by
- [FIELD] – See Query Operators, Valid Query Fields
- date_group – See Query Operators
- group_by – See Query Operators
- order_by – See Query Operators
- aggregate_by – See Query Operators
- date_range – See Query Operators
- limit – See Query Operators
- offset – See Query Operators

**Returns** 200 OK with a list of procedures, or 400 Bad Request if any invalid query parameters were passed.

Example Return Value:

```xml
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*" />
    <Filters>
      </Filters>
  </QueryParams>
  <Report>
    <Meta>
    <Document id="261ca370-927f-41af-b001-7b615c7a468e" type="http://indivo.org/vocab/xml/documents#">
      <createdAt>2011-05-02T17:48:13Z</createdAt>
      <creator id="mymail@mail.ma" type="Account">
        <fullname>full name</fullname>
      </creator>
      <original id="261ca370-927f-41af-b001-7b615c7a468e"/>
      <label>testing</label>
      <status>active</status>
      <nevershare>false</nevershare>
    </Document>
    </Meta>
    <Item>
    <Procedure xmlns="http://indivo.org/vocab/xml/documents#">
      <datePerformed>2009-05-16T12:00:00</datePerformed>
    </Procedure>
  </Item>
</Report>
</Reports>
```
Appendectomy

<provider>
  <name>Kenneth Mandl</name>
  <institution>Children’s Hospital Boston</institution>
</provider>

GET /carenets/{CARENET_ID}/reports/minimal/simple-clinical-notes/
List the simple_clinical_notes data for a given carenet.

  Short Name  carenet_simple_clinical_notes_list

  Accessible By  A user app with access to the carenet or the entire carenet’s record, or an account in
                   the carenet or in control of the record.

  URL Parameters
    • CARENET_ID  – The id string associated with the Indivo carenet

  Query Parameters
    • status  – The account or document status to filter by
    • {FIELD}  – See Query Operators, Valid Query Fields
    • date_group  – See Query Operators
    • group_by  – See Query Operators
    • order_by  – See Query Operators
    • aggregate_by  – See Query Operators
    • date_range  – See Query Operators
    • limit  – See Query Operators
    • offset  – See Query Operators

  Returns  200 OK with a list of notes, or 400 Bad Request if any invalid query parameters were
           passed.

Example Return Value:

<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*" />
    <Filters/>
  </QueryParams>
  <Report>
    <Meta>
      <Document id="261ca370-927f-41af-b001-7b615c7a468e" type="http://indivo.org/vocab/xml/documents#SimpleClinicalNote"
        createdAt="2011-05-02T17:48:13Z" />
      <creator id="mymail@mail.ma" type="Account">
        <createdAt>2011-05-02T17:48:13Z</createdAt>
      </creator>
    </Meta>
  </Report>
</Reports>
GET /carenets/{CARENET_ID}/reports/{DATA_MODEL}/
List the Model data for a given carenet.

Short Name  carenet_generic_list

Accessible By  A user app with access to the carenet or the entire carenet’s record, or an account in the carenet or in control of the record.

URL Parameters

• DATA_MODEL – The name of the data model to report on

• CARENET_ID – The id string associated with the Indivo carenet

Query Parameters

• status – The account or document status to filter by

• {FIELD} – See Query Operators, Valid Query Fields
• date_group – See Query Operators
• group_by – See Query Operators
• order_by – See Query Operators
• aggregate_by – See Query Operators
• date_range – See Query Operators
• limit – See Query Operators
• offset – See Query Operators
• response_format – See Response Formats

Returns 200 OK with a list of DATA_MODELs, or 400 Bad Request if any invalid query parameters were passed.

Example Return Value:

SDMX Example:
{
   "__modelname__": "TestMedication",
   "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
   "name": "ibuprofen",
   "date_started": "2010-10-01T00:00:00Z",
   "date_stopped": "2010-10-31T00:00:00Z",
   "brand_name": "Advil",
   "prescription": {
      "__modelname__": "TestPrescription",
      "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
      "prescribed_by_name": "Kenneth D. Mandl",
      "prescribed_by_institution": "Children’s Hospital Boston",
      "prescribed_on": "2010-09-30T00:00:00Z",
      "prescribed_stop_on": "2010-10-31T00:00:00Z"
   },
   "fills": [
      {
      "__modelname__": "TestFill",
      "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
      "date_filled": "2010-10-01T00:00:00Z",
      "supply_days": "15",
      "filled_at_name": "CVS"
      },
      {
      "__modelname__": "TestFill",
      "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
      "date_filled": "2010-10-16T00:00:00Z",
      "supply_days": "15",
      "filled_at_name": "CVS"
      }
   ]
}

SDMX Example:

<Models>
   <Model name="TestMedication" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
      <Field name="date_started">2010-10-01T00:00:00Z</Field>
      <Field name="name">ibuprofen</Field>
      <Field name="brand_name">Advil</Field>
   </Model>
</Models>
GET /codes/systems/
List available codingsystems. NOT IMPLEMENTED.

**Short Name**  coding_systems_list

**Accessible By**  Anybody

**Returns**  500 Internal Server Error, as the system cannot process the call.

Example Return Value:
```javascript
[{"short_name": "umls-snomed", "name": "UMLS SNOMED", "description" : "..."},
 (...),
 (...)]
```

**Todo**
The API Call ‘GET /codes/systems/’ is not yet implemented.

GET /codes/systems/{SYSTEM_SHORT_NAME}/query
Query a codingsystem for a value.

**Short Name**  coding_system_query

**Accessible By**  Anybody

**URL Parameters**
- SYSTEM_SHORT_NAME –

**Query Parameters**
- q – The query string to search for
Returns 200 OK with JSON describing codingsystems entries that matched q, or 404 Not Found if SYSTEM_SHORT_NAME is invalid.

Example Return Value:

```json
[{
  "abbreviation": null,
  "code": "38341003",
  "consumer_value": null,
  "umls_code": "C0020538",
  "full_value": "Hypertensive disorder, systemic arterial (disorder)"
},
{
  "abbreviation": null,
  "code": "55822004",
  "consumer_value": null,
  "umls_code": "C0020473",
  "full_value": "Hyperlipidemia (disorder)"
}]
```

**POST /oauth/access_token**

Exchange a request token for a valid access token.

**Short Name** exchange_token

**Accessible By** A request signed by a RequestToken.

**Returns** 200 OK with an access token, or 403 Forbidden if the request token didn’t validate.

Example Return Value:

```
oauth_token=abcd1fw3gasdgh3&oauth_token_secret=jgrlhre4291hfjas&oauth_indivo_record_id=123
```

**POST /oauth/internal/request_tokens/{REQTOKEN_ID}/approve**

Indicate a user’s consent to bind an app to a record or carenet.

**Short Name** request_token_approve

**Accessible By** A principal in the carenet to which the request token is restricted (if the token is restricted), or a principal with full control over the record (if the token is not restricted).

**URL Parameters**

- **REQTOKEN_ID**

**Formdata Parameters**

- **record_id** – The record to bind to. Either record_id or carenet_id is required.
- **carenet_id** – The carenet to bind to. Either record_id or carenet_id is required.

**Returns** 200 OK with a redirect url to the app on success, 403 Forbidden if record_id/carenet_id don’t match reqtoken.

Example Return Value:

```
location=http%3A%2F%2Fapps.indivo.org%2Fproblems%2Fafter_auth%3Foauth_token%3Dabc123%26oauth_verifier%3Dabc123
```

(which is the urlencoded form of:

```
http://apps.indivo.org/problems/after_auth?oauth_token=abc123&oauth_verifier=abc123
```

**POST /oauth/internal/request_tokens/{REQTOKEN_ID}/claim**

Claim a request token on behalf of an account.

**Short Name** request_token_claim

**Accessible By** Any Account.

**URL Parameters**
• REQTOKEN_ID –

Returns 200 OK with the email of the claiming principal, or 403 Forbidden if the token has already been claimed.

Example Return Value:

joeuser@indivo.org

GET /oauth/internal/request_tokens/{REQTOKEN_ID}/info
Get information about a request token.

Short Name request_token_info
Accessible By Any Account.

URL Parameters
• REQTOKEN_ID –

Returns 200 OK with information about the token.

Example Return Value:

<RequestToken token="XYZ">
  <record id="123" />  
  <carenet />  
  <kind>new</kind>  
  <App id="problems@apps.indivo.org">
    <name>Problem List</name>  
    <description>Managing your list of problems</description>  
    <autonomous>false</autonomous>  
    <frameable>true</frameable>  
    <ui>true</ui>
  </App>
</RequestToken>

POST /oauth/internal/session_create
Authenticate a user and register a web session for them.

Short Name session_create
Accessible By Any Indivo UI app.

Formdata Parameters
• username – The username of the user to authenticate.
• password – The password to use with username against the internal password auth system. EITHER password or system is Required.
• system – An external auth system to authenticate the user with. EITHER password or system is Required.

Returns 200 OK with a valid session token, or 403 Forbidden if the passed credentials were invalid.

Example Return Value:

oauth_token=XYZ&oauth_token_secret=ABC&account_id=joeuser%40indivo.org
GET /oauth/internal/surl-verify
Verify a signed URL.

Short Name  surl_verify
Accessible By  Any Account.

Query Parameters
•  surl_sig  – The computed signature (base-64 encoded sha1) of the url.
•  surl_timestamp  – when the url was generated. Must be within the past hour.
•  surl_token  – The access token used to sign the url.

Returns  200 OK with XML describing whether the surl validated.

Example Return Value:
If the surl validated:
<!--result>ok</result-->

If the surl was too old:
<!--result>old</result-->

If the surl’s signature was invalid:
<!--result>mismatch</result-->

POST /oauth/request_token
Get a new request token, bound to a record or carenet if desired.

Short Name  request_token
Accessible By  Any user app.

Formdata Parameters
•  indivo_record_id  – The record to which to bind the request token. EITHER indivo_record_id or indivo_carenet_id is REQUIRED.
•  indivo_carenet_id  – The carenet to which to bind the request token. EITHER indivo_record_id or indivo_carenet_id is REQUIRED.

Returns  200 OK with the request token on success, 403 Forbidden if the oauth signature on the request of missing or faulty.

Example Return Value:
oauth_token=abcd1fw3gasdgh3&oauth_token_secret=jgrlhre4291hfjassxoauth_indivo_record_id=123

GET /ontology
Fetch the SMART ontology as RDF/XML.

Short Name  smart_ontology
Accessible By  Any principal in Indivo.

Returns  An OWL file describing the SMART ontology.
Example Return Value:

see http://sandbox-api.smartplatforms.org/ontology

New in version 2.0.0.

POST /records/
Create a new record.

    Short Name     record_create
    Accessible By  Any admin app.
    Raw Data       A valid Indivo Demographics Document (see Indivo Document Demographics Schema).
    Returns        200 OK with information about the record on success, 400 Bad Request if the demographics XML was empty or invalid.

Example Return Value:

    <Record id="123" label="Joe Smith">
       <demographics document_id="" />
    </Record>

PUT /records/external/{PRINCIPAL_EMAIL}/{EXTERNAL_ID}
Create a new record with an associated external id.

    Short Name     record_create_ext
    Accessible By  An admin app with an id matching the principal_email in the URL.
    URL Parameters
        • PRINCIPAL_EMAIL – The email with which to scope an external id.
        • EXTERNAL_ID – The external identifier of the desired resource
    Raw Data       A valid Indivo Demographics Document (see Indivo Document Demographics Schema).
    Returns        200 OK with information about the record on success, 400 Bad Request if the demographics XML was empty or invalid.

Example Return Value:

    <Record id="123" label="Joe Smith">
       <demographics document_id="" />
    </Record>

GET /records/search
Search for records by label (usually the same as full name).

    Short Name     record_search
    Accessible By  Any admin app.
    Query Parameters
        • label – A search string to match against record labels.
    Returns        200 OK with a list of matching records on success, 400 Bad Request if no query parameters were passed.
Example Return Value:

```xml
<Records>
  <Record id="123" label="John R. Smith" />
  <Record id = "234" label="Frank Frankson" />
  ...
</Records>
```

New in version 1.0.1.

**GET /records/{RECORD_ID}**

Get information about an individual record.

**Short Name** record

**Accessible By** A principal in full control of the record, any admin app, or a user app with access to the record.

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record

**Returns** 200 OK with information about the record.

Example Return Value:

```xml
<Record id="123" label="Joe Smith">
  <demographics document_id="346" />
</Record>
```

**GET /records/{RECORD_ID}/allergies/**

SMART allergy list, serialized as RDF/XML.

**Short Name** smart_allergies

**Accessible By** A user app with access to the record, or a principal in full control of the record

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record

**Returns** SMART RDF describing the record’s allergies and allergy exclusions

Example Return Value:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:sp="http://smartplatforms.org/terms#"
>
  <rdf:Description rdf:about="http://purl.bioontology.org/ontology/SNOMEDCT/271807003">
    <dcterms:title>skin rash</dcterms:title>
    <dcterms:identifier>271807003</dcterms:identifier>
    <sp:system>http://purl.bioontology.org/ontology/SNOMEDCT</sp:system>
    <rdf:type rdf:resource="http://smartplatforms.org/terms/codes/SNOMED"/>
    <rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
  </rdf:Description>
</rdf:RDF>
```
<sp:category rdf:nodeID="_865481f6-03ca-4707-8a89-ec468952efa5"/>
<sp:severity rdf:nodeID="_9f6a6981-1173-4041-8fa3-4462238ab8ae"/>
<sp:foodAllergen rdf:nodeID="_24be52e0-51a4-4d00-9654-25ae9e0ad2f4"/>
<rdf:type rdf:resource="http://smartplatforms.org/terms#Allergy"/>
<sp:belongsTo rdf:nodeID="_f63e49ae-5071-4f99-a62d-329a2e23ce85"/>
<sp:allergicReaction rdf:nodeID="_7912ae70-da78-443f-a0b6-3f955b9e140a"/>
</rdf:Description>
<rdf:Description rdf:nodeID="_865481f6-03ca-4707-8a89-ec468952efa5">
<dcterms:title>food allergy</dcterms:title>
<rdf:type rdf:resource="http://smartplatforms.org/terms#CodedValue"/>
</rdf:Description>
<rdf:Description rdf:nodeID="_e6301747-0618-4a74-9b52-d7b5f3745463">
<dcterms:title>drug allergy</dcterms:title>
<rdf:type rdf:resource="http://smartplatforms.org/terms#CodedValue"/>
</rdf:Description>
<rdf:Description rdf:nodeID="_b8361d90-8b1e-40cb-b892-80db4301d90">
<dcterms:title>mild</dcterms:title>
<rdf:type rdf:resource="http://smartplatforms.org/terms#CodedValue"/>
</rdf:Description>
<rdf:Description rdf:about="http://purl.bioontology.org/ontology/SNOMEDCT/414285001">
<dcterms:title>food allergy</dcterms:title>
<dcterms:identifier>414285001</dcterms:identifier>
<sp:system>http://purl.bioontology.org/ontology/SNOMEDCT/</sp:system>
<rdf:type rdf:resource="http://smartplatforms.org/terms/codes/AllergyCategory"/>
<rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
</rdf:Description>
<rdf:Description rdf:nodeID="_7912ae70-da78-443f-a0b6-3f955b9e140a">
<dcterms:title>anaphylaxis</dcterms:title>
<rdf:type rdf:resource="http://smartplatforms.org/terms#CodedValue"/>
</rdf:Description>
<rdf:Description rdf:nodeID="_9f6a6981-1173-4041-8fa3-4462238ab8ae">
<dcterms:title>severe</dcterms:title>
<rdf:type rdf:resource="http://smartplatforms.org/terms#CodedValue"/>
</rdf:Description>
<rdf:Description rdf:about="http://purl.bioontology.org/ontology/SNOMEDCT/416098002">
<dcterms:title>drug allergy</dcterms:title>
<dcterms:identifier>416098002</dcterms:identifier>
<sp:system>http://purl.bioontology.org/ontology/SNOMEDCT/</sp:system>
<rdf:type rdf:resource="http://smartplatforms.org/terms/codes/AllergyCategory"/>
<rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
</rdf:Description>
<rdf:Description rdf:nodeID="_2c0973ad-d7fd-4e7f-a6c2-5c625a54aae7">
<dcterms:title>skin rash</dcterms:title>
<rdf:type rdf:resource="http://smartplatforms.org/terms#CodedValue"/>
</rdf:Description>
<rdf:Description rdf:about="http://indivo.org/records/03059111-af61-4834-8234-befe5f5a2532/allergies/a4abf13b-ec73-4ae4-b0a2-9d71ab2ea368">
<sp:drugClassAllergen rdf:nodeID="_f06759ab-6668-4832-97da-0794d244d403"/>
<sp:category rdf:nodeID="_e6301747-0618-4a74-9b52-d7b5f3745463"/>
<sp:severity rdf:nodeID="_b8361d90-8b1e-40cb-b892-80db4301d90"/>
<rdf:type rdf:resource="http://smartplatforms.org/terms#Allergy"/>
<sp:belongsTo rdf:nodeID="_f63e49ae-5071-4f99-a62d-329a2e23ce85"/>
<sp:allergicReaction rdf:nodeID="_2c0973ad-d7fd-4e7f-a6c2-5c625a54aae7"/>
New in version 2.0.0.

GET /records/{RECORD_ID}/allergies/{MODEL_ID}
Retrieve a specific instance of a SMART Allergy/AllergyExclusion.

Short Name smart_allergies_instance
Accessible By  A user app with access to the record, or a principal in full control of the record

URL Parameters
• RECORD_ID – The id string associated with the Indivo record
• MODEL_ID – The id of the Allergy/AllergyExclusion

Returns  SMART RDF describing the Allergy/AllergyExclusion

Example Return Value:

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sp="http://smartplatforms.org/terms#"
>
  <rdf:Description rdf:nodeID="_9ea3b5f3-c0c4-40d7-a370-2ab475ea1d6e">
    <rdf:type rdf:resource="http://smartplatforms.org/terms#CodedValue"/>
    <dcterms:title>Sulfonamide Antibacterial</dcterms:title>
  </rdf:Description>
</rdf:RDF>

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sp="http://smartplatforms.org/terms#"
>
  <rdf:Description rdf:about="http://purl.bioontology.org/ontology/SNOMEDCT/416098002">
    <rdf:type rdf:resource="http://smartplatforms.org/terms/codes/AllergyCategory"/>
    <rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
    <dcterms:identifier>416098002</dcterms:identifier>
    <dcterms:title>Drug allergy</dcterms:title>
    <sp:system>http://purl.bioontology.org/ontology/SNOMEDCT/</sp:system>
  </rdf:Description>
</rdf:RDF>

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sp="http://smartplatforms.org/terms#"
>
  <rdf:Description rdf:about="http://purl.bioontology.org/ontology/SNOMEDCT/24484000">
    <rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
    <dcterms:identifier>24484000</dcterms:identifier>
    <dcterms:title>Severe</dcterms:title>
    <sp:system>http://purl.bioontology.org/ontology/SNOMEDCT/</sp:system>
  </rdf:Description>
</rdf:RDF>

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sp="http://smartplatforms.org/terms#"
>
  <rdf:Description rdf:about="http://purl.bioontology.org/ontology/SNOMEDCT/39579001">
    <rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
    <dcterms:identifier>39579001</dcterms:identifier>
    <dcterms:title>Anaphylaxis</dcterms:title>
    <sp:system>http://purl.bioontology.org/ontology/SNOMEDCT/</sp:system>
  </rdf:Description>
</rdf:RDF>

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sp="http://smartplatforms.org/terms#"
>
  <rdf:Description rdf:about="http://indivo.org/records/b171cd33-00a6-4038-976d-b8380c276ba1/allergies/09eadb0d-9c58-4cac-aad5-c84c29caf5bd">
    <rdf:type rdf:resource="http://smartplatforms.org/terms#Allergy"/>
    <sp:allergicReaction rdf:nodeID="_6538190b-0659-4710-b083-fe3f0462242b"/>
    <sp:category rdf:nodeID="_3572ccd-7787-4801-b6b3-e153794ace84"/>
    <sp:drugClassAllergen rdf:nodeID="_9ea3b5f3-c0c4-40d7-a370-2ab475ea1d6e"/>
    <sp:severity rdf:nodeID="_e62b76fe-e33b-4d9f-9a2b-dc1221e3eb17"/>
    <sp:belongsTo rdf:resource="http://indivo.org/records/b171cd33-00a6-4038-976d-b8380c276ba1"/>
  </rdf:Description>
</rdf:RDF>

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sp="http://smartplatforms.org/terms#"
>
  <rdf:Description rdf:nodeID="_6538190b-0659-4710-b083-fe3f0462242b">
    <dcterms:title>Severe</dcterms:title>
  </rdf:Description>
</rdf:RDF>

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sp="http://smartplatforms.org/terms#"
>
  <rdf:Description rdf:nodeID="_6538190b-0659-4710-b083-fe3f0462242b">
    <dcterms:title>Anaphylaxis</dcterms:title>
  </rdf:Description>
</rdf:RDF>

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sp="http://smartplatforms.org/terms#"
>
  <rdf:Description rdf:nodeID="_6538190b-0659-4710-b083-fe3f0462242b">
    <dcterms:title>Anaphylaxis</dcterms:title>
  </rdf:Description>
</rdf:RDF>
GET /records/{RECORD_ID}/apps/
List userapps bound to a given record.

**Short Name** record_phas

**Accessible By** A principal in full control of the record, or any admin app.

**URL Parameters**
- RECORD_ID – The id string associated with the Indivo record

**Query Parameters**
- type – A namespaced document type. If specified, only apps which explicitly declare themselves as supporting that document type will be returned.

**Returns** 200 OK with a list of JSON manifests for the userapps.

Example Return Value:

```json
[
 {
 "name": "SMART Problems",
 "description": "Display problems in a table view",
 "author": "Josh Mandel, Children’s Hospital Boston",
 "id": "problem-list@apps.smartplatforms.org",
 "version": ".1a",

 "mode": "ui",
 "scope": "record",

 "icon": "http://fda.gping.org:8012/framework/problem_list/icon.png",

 "requires": {
  "http://smartplatforms.org/terms#Problem": {
   "methods": ["GET"]
  }
 }
 },
...
other apps ...
```
Changed in version 2.0.0: Apps are now returned as JSON manifests, not XML.

DELETE /records/{RECORD_ID}/apps/{PHA_EMAIL}
Remove a userapp from a record.

  **Short Name** pha_record_delete
  **Accessible By** Any admin app, or a principal in full control of the record.

  **URL Parameters**
  - **RECORD_ID** – The id string associated with the Indivo record
  - **PHA_EMAIL** – The email identifier of the Indivo user app

  **Returns** 200 Success.

Example Return Value:

```xml
<ok/>
```

GET /records/{RECORD_ID}/apps/{PHA_EMAIL}
Get information about a given userapp bound to a record.

  **Short Name** record_pha
  **Accessible By** A principal in full control of the record, or any admin app.

  **URL Parameters**
  - **RECORD_ID** – The id string associated with the Indivo record
  - **PHA_EMAIL** – The email identifier of the Indivo user app

  **Returns** 200 OK with a JSON manifest for the app, or 404 Not Found if the app isn’t bound to the record.

Example Return Value:

```json
{
  "name" : "SMART Problems",
  "description" : "Display problems in a table view",
  "author" : "Josh Mandel, Children’s Hospital Boston",
  "id" : "problem-list@apps.smartplatforms.org",
  "version" : ".1a",
  "mode" : "ui",
  "scope" : "record",
  "requires" : {
    "http://smartplatforms.org/terms#Problem": {
      "methods": ["GET"]
    }
  }
}
```
Changed in version 2.0.0: Apps are now returned as JSON manifests, not XML

**PUT /records/{RECORD_ID}/apps/{PHA_EMAIL}**

Enable a userapp for a record.

**Short Name** record_pha_enable

**Accessible By** Any admin app, or a principal in full control of the record.

**URL Parameters**
- **RECORD_ID** – The id string associated with the Indivo record
- **PHA_EMAIL** – The email identifier of the Indivo user app

**Returns** 200 OK on success, 404 Not Found if either the specified record or the specified app doesn’t exist.

Example Return Value:

```
<ok/>
```

New in version 1.0.0.

**GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/**

List record-app-specific documents.

**Short Name** record_app_document_list

**Accessible By** A user app with access to the record, with an id matching the app email in the URL.

**URL Parameters**
- **RECORD_ID** – The id string associated with the Indivo record
- **PHA_EMAIL** – The email identifier of the Indivo user app

**Query Parameters**
- **status** – The account or document status to filter by
- **order_by** – See Query Operators
- **type** – The Indivo document type to filter by
- **limit** – See Query Operators
- **offset** – See Query Operators

**Returns** 200 OK with a list of documents, or 404 Not Found if an invalid type was passed in the querystring.

Example Return Value:

```
<Documents record_id="123" total_document_count="4" pha="problems@apps.indivo.org">
  <Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
    <createdAt>2009-05-04T17:05:33</createdAt>
    <creator id="steve@indivo.org" type="account">
      <fullname>Steve Zabak</fullname>
    </creator>
    <suppressedAt>2009-05-06T17:05:33</suppressedAt>
    <suppressor id="steve@indivo.org" type="account">
      <fullname>Steve Zabak</fullname>
    </suppressor>
  </Document>
</Documents>
```
POST /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/

Create a record-app-specific Indivo document.

**Short Name** record_app_document_create

**Accessible By** A user app with access to the record, with an id matching the app email in the URL.

**URL Parameters**
- RECORD_ID – The id string associated with the Indivo record
- PHA_EMAIL – The email identifier of the Indivo user app

**Raw Data** The raw content of the document to create.

**Returns** 200 OK with the metadata of the created document, or 400 Bad Request if the new document failed validation.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d54" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
    <label>HBA1C reading Preferences</label>
    <status>active</status>
    <nevershare>false</nevershare>
    <relatesTo>
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
      <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
    </relatesTo>
    <isRelatedFrom>
      <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
    </isRelatedFrom>
  </latest>
</Document>
```
**POST /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID}**

Create or Overwrite a record-app-specific Indivo document with an associated external id.

**Short Name** record_app_document_create_or_update_ext

**Accessible By** A user app with access to the record, with an id matching the app email in the URL.

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record
- **EXTERNAL_ID** – The external identifier of the desired resource
- **PHA_EMAIL** – The email identifier of the Indivo user app

**Raw Data** The raw content of the document to create/update.

**Returns** 200 OK with metadata describing the created or updated document, or 400 Bad Request if the passed content didn’t validate.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="problems@apps.indivo.org" type="pha">
  </creator>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" status="active"
  <label>HBA1C reading preferences</label>
  <nevershare>false</nevershare>
</Document>
```

**PUT /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID}**

Create or Overwrite a record-app-specific Indivo document with an associated external id.

**Short Name** record_app_document_create_or_update_ext

**Accessible By** A user app with access to the record, with an id matching the app email in the URL.

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record
- **EXTERNAL_ID** – The external identifier of the desired resource
- **PHA_EMAIL** – The email identifier of the Indivo user app

**Raw Data** The raw content of the document to create/update.

**Returns** 200 OK with metadata describing the created or updated document, or 400 Bad Request if the passed content didn’t validate.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="problems@apps.indivo.org" type="pha">
  </creator>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" status="active">
  </latest>
</Document>
```
GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID}/meta
Fetch the metadata of a record-app-specific document identified by external id.

**Short Name** record_app_document_meta_ext

**Accessible By** A user app with access to the record, with an id matching the app email in the URL.

**URL Parameters**
- **RECORD_ID** – The id string associated with the Indivo record
- **EXTERNAL_ID** – The external identifier of the desired resource
- **PHA_EMAIL** – The email identifier of the Indivo user app

**Returns** 200 OK with metadata describing the specified document, or http:statuscode:404 if the external_id is invalid.

Example Return Value:

```html
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77"
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="problems@apps.indivo.org" type="pha">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" 
    <label>HBA1C reading preferences</label>
    <status>active</status>
    <nevershare>false</nevershare>
  </latest>
</Document>
```

DELETE /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}
Delete a record-app-specific document.

**Short Name** record_app_document_delete

**Accessible By** A user app with access to the record, with an id matching the app email in the URL.

**URL Parameters**
- **RECORD_ID** – The id string associated with the Indivo record
- **PHA_EMAIL** – The email identifier of the Indivo user app
- **DOCUMENT_ID** – The unique identifier of the Indivo document

**Returns** 200 Success, or 404 Not Found if DOCUMENT_ID is invalid.

Example Return Value:
GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}
Retrieve a record-app-specific document.

Short Name  record_app_specific_document
Accessible By  A user app with access to the record, with an id matching the app email in the URL.

URL Parameters
- RECORD_ID – The id string associated with the Indivo record
- PHA_EMAIL – The email identifier of the Indivo user app
- DOCUMENT_ID – The unique identifier of the Indivo document

Returns 200 OK with the raw content of the document, or 404 Not Found if the document could not be found.

Example Return Value:

```xml
<ProblemsPreferences record_id="123">
  <Preference name="hide_void" value="true" />
  <Preference name="show_rels" value="false" />
</ProblemsPreferences>
```

PUT /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/label
Set the label of a record-app-specific document.

Short Name  record_app_document_label
Accessible By  A user app with access to the record, with an id matching the app email in the URL.

URL Parameters
- RECORD_ID – The id string associated with the Indivo record
- PHA_EMAIL – The email identifier of the Indivo user app
- DOCUMENT_ID – The unique identifier of the Indivo document

Raw Data  The new label for the document

Returns 200 OK with metadata describing the re-labeled document, or 404 Not Found if DOCUMENT_ID is invalid.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d54" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org"/>
  <label>RELABELED: New HBA1C reading Preferences</label>
</Document>
```
GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/meta
Fetch the metadata of a record-app-specific document.

Short Name record_app_document_meta
Accessible By A user app with access to the record, with an id matching the app email in the URL.

URL Parameters
- RECORD_ID – The id string associated with the Indivo record
- PHA_EMAIL – The email identifier of the Indivo user app
- DOCUMENT_ID – The unique identifier of the Indivo document

Returns 200 OK with the document metadata, or 404 Not Found if DOCUMENT_ID is invalid.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" label="HBA1C reading Preferences" status="active" nevershare="false">
    <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
  </latest>
  <label>HBA1C reading Preferences</label>
  <status>active</status>
  <nevershare>false</nevershare>
  <relatesTo>
    <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
  </relatesTo>
  <isRelatedFrom>
    <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
  </isRelatedFrom>
</Document>
```

POST /records/{RECORD_ID}/apps/{PHA_EMAIL}/setup
Bind an app to a record without user authorization.

Short Name record_pha_setup
Accessible By Any admin app.
**URL Parameters**
- **RECORD_ID** – The id string associated with the Indivo record
- **PHA_EMAIL** – The email identifier of the Indivo user app

**Raw Data** Raw content that will be used as a setup document for the record. **OPTIONAL.**

**Returns** 200 OK with a valid access token for the newly set up app.

Example Return Value:
```
oauth_token=abcd1fw3gasdgh3&oauth_token_secret=jgrlhre4291hfjas&xoauth_indivo_record_id=123
```

**GET /records/{RECORD_ID}/audits/**
Return audits of calls touching record.

**Short Name** audit_record_view

**Accessible By** A principal in full control of the record, or a user app with access to the record.

**URL Parameters**
- **RECORD_ID** – The id string associated with the Indivo record

**Query Parameters**
- **limit** – See Query Operators
- **order_by** – See Query Operators
- **offset** – See Query Operators

**Returns** 200 OK, with a list of Audit Reports.

Example Return Value:
```
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <Filters/>
  </QueryParams>
  <Report>
    <Meta/>
    <Item>
      <AuditEntry>
        <BasicInfo datetime="2011-04-27T17:32:23Z" view_func="get_document" request_successful="true"/>
        <PrincipalInfo effective_principal="myapp@apps.indivohealth.org" proxied_principal="me@indivohealth.org"/>
        <Resources carenet_id="" record_id="123" pha_id="" document_id="234" external_id="" message_id=""/>
        <RequestInfo req_url="/records/123/documents/acd/" req_ip_address="127.0.0.1" req_domain="localhost" req_method="GET"/>
        <ResponseInfo resp_code="200"/>
      </AuditEntry>
    </Item>
  </Report>

  ...
</Reports>
```

Deprecated since version 0.9.3: Use **GET /records/{RECORD_ID}/audits/query/** instead.
GET /records/{RECORD_ID}/audits/documents/{DOCUMENT_ID}/
Return audits of calls touching record and document_id.

**Short Name**  audit_document_view

**Accessible By**  A principal in full control of the record, or a user app with access to the record.

**URL Parameters**
- **RECORD_ID**  – The id string associated with the Indivo record
- **DOCUMENT_ID**  – The unique identifier of the Indivo document

**Query Parameters**
- **limit**  – See Query Operators
- **order_by**  – See Query Operators
- **offset**  – See Query Operators

**Returns**  200 OK, with a list of Audit Reports.

Example Return Value:

```xml
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <Filters>
      <Filter name="document_id" value="234"/>
    </Filters>
  </QueryParams>
  <Report>
    <Meta>
    </Meta>
    <Item>
      <AuditEntry>
        <BasicInfo datetime="2011-04-27T17:32:23Z" view_func="get_document" request_successful="true" />
        <PrincipalInfo effective_principal="myapp@apps.indivohealth.org" proxied_principal="me@indivohealth.org" />
        <Resources carenet_id="" record_id="123" pha_id="" document_id="234" external_id="" message_id="" />
        <RequestInfo req_url="/records/123/documents/acd/" req_ip_address="127.0.0.1" req_domain="localhost" req_method="GET" />
        <ResponseInfo resp_code="200" />
      </AuditEntry>
    </Item>
  </Report>
...
</Reports>
```

Deprecated since version 0.9.3: Use GET /records/{RECORD_ID}/audits/query/ instead.

GET /records/{RECORD_ID}/audits/documents/{DOCUMENT_ID}/functions/{FUNCTION_NAME}/
Return audits of calls to function_name touching record and document_id.

**Short Name**  audit_function_view

**Accessible By**  A principal in full control of the record, or a user app with access to the record.

**URL Parameters**
- **RECORD_ID**  – The id string associated with the Indivo record
• DOCUMENT_ID – The unique identifier of the Indivo document

• FUNCTION_NAME – The internal Indivo function name called by the API request

Query Parameters

• limit – See Query Operators

• order_by – See Query Operators

• offset – See Query Operators

Returns 200 OK, with a list of Audit Reports.

Example Return Value:

```xml
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <Filters>
      <Filter name="document_id" value="234"/>
      <Filter name="req_view_func" value="record_specific_document"/>
    </Filters>
  </QueryParams>
  <Report>
    <Meta>
    </Meta>
    <Item>
      <AuditEntry>
        <BasicInfo datetime="2011-04-27T17:32:23Z" view_func="get_document" request_successful="true"/>
        <PrincipalInfo effective_principal="myapp@apps.indivohealth.org" proxied_principal="me@indivohealth.org"/>
        <Resources carenet_id="" record_id="123" pha_id="" document_id="234" external_id="" message_id=""/>
        <RequestInfo req_url="/records/123/documents/acd/" req_ip_address="127.0.0.1" req_domain="localhost" req_method="GET"/>
        <ResponseInfo resp_code="200"/>
      </AuditEntry>
    </Item>
  </Report>
  ...
</Reports>
```

Deprecated since version 0.9.3: Use GET /records/{RECORD_ID}/audits/query/ instead.

GET /records/{RECORD_ID}/audits/query/

Select Audit Objects via the Query API Interface.

Short Name audit_query

Accessible By A principal in full control of the record, or a user app with access to the record.

URL Parameters

• RECORD_ID – The id string associated with the Indivo record

Query Parameters

• status – The account or document status to filter by

• {FIELD} – See Query Operators, Querying the Audit System

• date_group – See Query Operators
Indivo X Documentation, Release 2.0.0

- `group_by` – See Query Operators
- `order_by` – See Query Operators
- `aggregate_by` – See Query Operators
- `date_range` – See Query Operators
- `limit` – See Query Operators
- `offset` – See Query Operators

Returns 200 OK with a list of audit records, or 400 Bad Request if any of the arguments to the query interface are invalid.

Example Return Value:

```
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="created_at*1995-03-10T00:00:00Z*" />
    <Filters>
      <Filter name="document_id" value="234"/>
    </Filters>
  </QueryParams>
  <Report>
    <Meta>
    </Meta>
    <Item>
      <AuditEntry>
        <BasicInfo datetime="2011-04-27T17:32:23Z" view_func="get_document" request_successful="true"/>
        <PrincipalInfo effective_principal="myapp@apps.indivohealth.org" proxied_principal="me@indivohealth.org"/>
        <Resources carenet_id="" record_id="123" pha_id="" document_id="234" external_id="" message_id=""/>
        <RequestInfo req_url="/records/123/documents/acd/" req_ip_address="127.0.0.1" req_domain="localhost" req_method="GET"/>
        <ResponseInfo resp_code="200"/>
      </AuditEntry>
    </Item>
  </Report>
  ...
</Reports>
```

New in version 0.9.3.

GET /records/{RECORD_ID}/autoshare/bytype/  
For a single record, list all carenets that a given doctype is autoshared with.

**Short Name** autoshare_list

**Accessible By** A principal in full control of the record.

**URL Parameters**

- `RECORD_ID` – The id string associated with the Indivo record

**Query Parameters**

- `type` – The document schema type to check autoshares for. **REQUIRED.**

Returns 200 OK with a list of carenets, or 404 Not Found if the passed document type is invalid.

Example Return Value:
GET /records/{RECORD_ID}/autoshare/bytype/all
For a single record, list all doctypes autoshared into carenets.

| Short Name | autoshare_list_bytype_all |
| Accessible By | A principal in full control of the record. |
| URL Parameters | |
| RECORD_ID – The id string associated with the Indivo record |

Returns 200 OK with a list of doctypes and their shared carenets.

Example Return Value:

```
<DocumentSchemas>
  <DocumentSchema type="http://indivo.org/vocab/xml/documents#Medication">
    <Carenet id="123" name="Family" mode="explicit" />
  ...
  </DocumentSchema>
...
</DocumentSchemas>
```

POST /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/set
Automatically share all documents of a certain type into a carenet.

| Short Name | autoshare_create |
| Accessible By | A principal in full control of the record. |
| URL Parameters | |
| RECORD_ID – The id string associated with the Indivo record |
| CARENET_ID – The id string associated with the Indivo carenet |

Formdata Parameters

| type – the document schema type to create an autoshare for |

Returns 200 OK, or 404 Not Found if the passed document type doesn’t exist.

Example Return Value:

```
<ok/>
```

POST /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/unset
Remove an autoshare from a carenet.
Short Name  autoshare_delete
Accessible By  A principal in full control of the record.

URL Parameters
- RECORD_ID – The id string associated with the Indivo record
- CARENET_ID – The id string associated with the Indivo carenet

Formdata Parameters
- type – the document schema type to remove an autoshare for

Returns 200 OK, or 404 Not Found if the passed document type doesn’t exist.

Example Return Value:

```xml
<ok/>
```

---

GET /records/{RECORD_ID}/carenets/
List all carenets for a record.

Short Name  carenet_list
Accessible By  A principal in full control of the record, or any admin app.

URL Parameters
- RECORD_ID – The id string associated with the Indivo record

Returns 200 OK, with a list of carenets.

Example Return Value:

```xml
<Carenets record_id="123">
  <Carenet id="789" name="Work/School" mode="explicit"/>
  ...
</Carenets>
```

---

POST /records/{RECORD_ID}/carenets/
Create a new carenet for a record.

Short Name  carenet_create
Accessible By  A principal in full control of the record, or any admin app.

URL Parameters
- RECORD_ID – The id string associated with the Indivo record

Formdata Parameters
- name – The label for the new carenet.

Returns 200 OK with a description of the new carenet, or 400 Bad Request if the name of the carenet wasn’t passed or already exists.

Example Return Value:
GET /records/{RECORD_ID}/demographics
Read demographics from a record.

Short Name  read_demographics

Accessible By  A user app with access to the record, a principal in full control of the record, or any
admin app.

URL Parameters

- RECORD_ID – The id string associated with the Indivo record

Query Parameters

- response_format – one of application/rdf+xml (SMART RDF/XML), application/json (SDMJ), or application/xml (SDMX). Default is application/rdf+xml

Returns  200 OK with demographics formatted in the requested response_format (default application/rdf+xml) on success, 404 Not Found when no demographics found, and 400 Bad Request if response_format is invalid

Example Return Value:

application/rdf+xml:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:dcterms="http://purl.org/dc/terms/"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:sp="http://smartplatforms.org/terms#"
    xmlns:v="http://www.w3.org/2006/vcard/ns#">
    <rdf:Description rdf:nodeID="_6730841b-05df-445f-8695-ed64197f4e6a">
        <v:family-name>William</v:family-name>
        <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Name"/>
        <v:given-name>Robinson</v:given-name>
    </rdf:Description>
    <rdf:Description rdf:nodeID="_bcf66b59-e438-49b2-b572-99af4319b297">
        <rdf:value>800-870-3011</rdf:value>
        <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Tel"/>
        <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Home"/>
        <rdf:type rdf:resource="http://www.w3.org/2006/vcard/ns#Pref"/>
    </rdf:Description>
    <rdf:Description rdf:nodeID="_8cbe3da1-fb53-4d31-80b6-19e0d04220ad">
        <dcterms:identifier>http://indivo.org/records/96ff9eb2-3b18-4a0e-9df8-5a731b96d5d6</dcterms:identifier>
        <sp:system>Indivo Record</sp:system>
        <dcterms:title>Indivo Record 96ff9eb2-3b18-4a0e-9df8-5a731b96d5d6</dcterms:title>
        <rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
    </rdf:Description>
    <rdf:Description rdf:nodeID="_451ade87-b519-4c92-8d07-2bbfcb23999c">
        <dcterms:identifier>http://indivo.org/records/96ff9eb2-3b18-4a0e-9df8-5a731b96d5d6</dcterms:identifier>
        <sp:system>Indivo Record</sp:system>
        <dcterms:title>Indivo Record 96ff9eb2-3b18-4a0e-9df8-5a731b96d5d6</dcterms:title>
        <rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
    </rdf:Description>
</rdf:RDF>
```
"__modelname__": "Demographics",
"__documentid__": "44190967-cbaa-43a7-a98c-9f97f094ef2b",
"bday": "1965-08-09",
"email": "william.robinson@example.com",
"gender": "male",
"preferred_language": "EN",
"name_given": "Robinson",
"name_family": "William",
"tel_2_type": "h",
"tel_2_preference": true,
"tel_2_number": "800-870-3011",
"adr_region": "OK",
"adr_country": "USA",
"adr_postalcode": "74008",
"adr_city": "Bixby",
"adr_street": "23 Church Rd",
"tel_1_type": "h",
"tel_1_preference": true,
"tel_1_number": "800-870-3011"}
"
PUT /records/{RECORD_ID}/demographics
Create or update demographics on a record.

**Short Name**  set_demographics  

**Accessible By**  A user app with access to the record, a principal in full control of the record, or any admin app.

**URL Parameters**
- **RECORD_ID** - The id string associated with the Indivo record

**Returns**  200 OK with metadata on the updated document, or 400 Bad Request if the new content didn’t validate

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="http://indivo.org/vocab/xml/documents#Demographics" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
<createdAt>2009-05-04T17:05:33</createdAt>
<creator id="steve@indivo.org" type="account">
<fullname>Steve Zabak</fullname>
</creator>
<original id="14c81023-c84f-496d-8b8e-9438280441d3" />
<latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" status="active">
<nevershare>false</nevershare>
</latest>
</Document>
```

New in version 2.0.0.

DELETE /records/{RECORD_ID}/documents/
Delete all documents associated with a record.
Indivo X Documentation, Release 2.0.0

Short Name documents_delete
Accessible By Nobody

URL Parameters
• RECORD_ID – The id string associated with the Indivo record

Returns 200 Success

Example Return Value:
<ok/>

GET /records/{RECORD_ID}/documents/
List record-specific documents.

Short Name record_document_list
Accessible By A user app with access to the record, or a principal in full control of the record

URL Parameters
• RECORD_ID – The id string associated with the Indivo record

Query Parameters
• status – The account or document status to filter by
• order_by – See Query Operators
• type – The Indivo document type to filter by
• limit – See Query Operators
• offset – See Query Operators

Returns 200 OK with a list of documents, or 404 Not Found if an invalid type was passed in the querystring.

Example Return Value:
<Documents record_id="123" total_document_count="4">
  <Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d4" createdAt="2009-05-04T17:05:33" suppressedAt="2009-05-06T17:05:33" createdBy="steve@indivo.org" suppressor="steve@indivo.org">
    <createdAt>2009-05-04T17:05:33</createdAt>
    <creator id="steve@indivo.org" type="account">
      <fullname>Steve Zabak</fullname>
    </creator>
    <suppressedAt>2009-05-06T17:05:33</suppressedAt>
    <suppressor id="steve@indivo.org" type="account">
      <fullname>Steve Zabak</fullname>
    </suppressor>
    <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
    <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
      <label>HBA1C reading</label>
      <status>active</status>
      <nevershare>false</nevershare>
      <relatesTo>
        <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
        <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
      </relatesTo>
      <isRelatedFrom>
        <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
      </isRelatedFrom>
    </latest>
  </Document>
  ...
</Documents>
POST /records/{RECORD_ID}/documents/
Create a record-specific Indivo Document.

Short Name  document_create

Accessible By  A user app with access to the record, a principal in full control of the record, or the admin app that created the record.

URL Parameters

- RECORD_ID – The id string associated with the Indivo record

Raw Data  The raw content of the document to create.

Returns 200 OK with the metadata of the created document, or 400 Bad Request if the new document failed validation.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d5d4" createdAt="2009-05-04T17:05:33" creator="steve@indivo.org" type="account">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
    <label>HBA1C reading</label>
    <status>active</status>
    <nevershare>false</nevershare>
    <relatesTo>
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
      <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
    </relatesTo>
    <isRelatedFrom>
      <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
    </isRelatedFrom>
  </latest>
</Document>
```

PUT /records/{RECORD_ID}/documents/external/{PHA_EMAIL}/{EXTERNAL_ID}
Create a record-specific Indivo Document with an associated external id.

Short Name  document_create_by_ext_id

Accessible By  A user app with access to the record, with an id matching the app email in the URL.

URL Parameters

- RECORD_ID – The id string associated with the Indivo record
- EXTERNAL_ID – The external identifier of the desired resource
• **PHA_EMAIL** – The email identifier of the Indivo user app

**Raw Data** The raw content of the document to create.

**Returns** 200 OK with the metadata of the created document, or 400 Bad Request if the new document failed validation, or if the external id was taken.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-Bb8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d5" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-Bb8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-Bb8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
    <label>HBA1C reading</label>
    <status>active</status>
    <nevershare>false</nevershare>
    <relatesTo>
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
      <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
    </relatesTo>
    <isRelatedFrom>
      <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
    </isRelatedFrom>
  </latest>
</Document>
```

**PUT /records/{RECORD_ID}/documents/external/{PHA_EMAIL}/{EXTERNAL_ID}/label**

Set the label of a record-specific document, specified by external id.

**Short Name** record_document_label_ext

**Accessible By** A user app with access to the record, with an id matching the app email in the URL.

**URL Parameters**

• **RECORD_ID** – The id string associated with the Indivo record

• **EXTERNAL_ID** – The external identifier of the desired resource

• **PHA_EMAIL** – The email identifier of the Indivo user app

**Raw Data** The new label for the document

**Returns** 200 OK with metadata describing the re-labeled document, or 404 Not Found if EXTERNAL_ID is invalid.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-Bb8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d5" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-Bb8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-Bb8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
    <label>HBA1C reading</label>
    <status>active</status>
    <nevershare>false</nevershare>
    <relatesTo>
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
      <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
    </relatesTo>
    <isRelatedFrom>
      <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
    </isRelatedFrom>
  </latest>
</Document>
```
GET /records/{RECORD_ID}/documents/external/{PHA_EMAIL}/{EXTERNAL_ID}/meta

Fetch the metadata of a record-specific document identified by external id.

Short Name  record_document_meta_ext

Accessible By  A user app with access to the record, with an id matching the app email in the URL.

URL Parameters

• RECORD_ID – The id string associated with the Indivo record
• EXTERNAL_ID – The external identifier of the desired resource
• PHA_EMAIL – The email identifier of the Indivo user app

Returns 200 OK with the document metadata, or 404 Not Found if EXTERNAL_ID is invalid.

Example Return Value:

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID_0}/rels/{REL}/{DOCUMENT_ID_1}

Create a new relationship between two existing documents.
Short Name  document_rels
Accessible By  A user app with access to the record, or a principal in full control of the record

URL Parameters
  • RECORD_ID  – The id string associated with the Indivo record
  • DOCUMENT_ID_1  – The id of the document that is the subject of the relationship, i.e. DOCUMENT_ID_1 annotates DOCUMENT_ID_0
  • DOCUMENT_ID_0  – The id of the document that is the object of the relationship, i.e. DOCUMENT_ID_0 is annotated by DOCUMENT_ID_1
  • REL  – The type of relationship between the documents, i.e. annotation, interpretation

Returns  200 Success, or 404 Not Found if DOCUMENT_ID_0, DOCUMENT_ID_1, or REL don’t exist.

Example Return Value:
<ok/>

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}
Retrieve a record-specific document.

Short Name  record_specific_document
Accessible By  A user app with access to the record, or a principal in full control of the record

URL Parameters
  • RECORD_ID  – The id string associated with the Indivo record
  • DOCUMENT_ID  – The unique identifier of the Indivo document

Returns  200 OK with the raw content of the document, or 404 Not Found if the document could not be found.

Example Return Value:
<HBA1C xmlns="http://indivo.org/vocab#" value="5.3" unit="percent" datetime="2011-01-15T17:00:00.000Z" />

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/
List all the carenets into which a document has been shared.

Short Name  document_carenets
Accessible By  A user app with access to the record, or a principal in full control of the record

URL Parameters
  • RECORD_ID  – The id string associated with the Indivo record
  • DOCUMENT_ID  – The unique identifier of the Indivo document

Returns  200 OK with a list of carenets.

Example Return Value:
DELETE /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}
Unshare a document from a given carenet.

   Short Name  carenet_document_delete
   Accessible By  A principal in full control of the carenet’s record.

   URL Parameters
   • RECORD_ID – The id string associated with the Indivo record
   • CARENET_ID – The id string associated with the Indivo carenet
   • DOCUMENT_ID – The unique identifier of the Indivo document

   Returns  200 Success, or 404 Not Found if DOCUMENT_ID is invalid or if either the passed carenet or document do not belong to the passed record.

Example Return Value:
<ok/>

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}
Place a document into a given carenet.

   Short Name  carenet_document_placement
   Accessible By  A principal in full control of the carenet’s record.

   URL Parameters
   • RECORD_ID – The id string associated with the Indivo record
   • CARENET_ID – The id string associated with the Indivo carenet
   • DOCUMENT_ID – The unique identifier of the Indivo document

   Returns  200 Success, or 404 Not Found if DOCUMENT_ID is invalid or nevershared.

Example Return Value:
<ok/>

POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}/autoshare-revert
Revert the document-sharing of a document in a carent to whatever rules are specified by autoshares. NOT IMPLEMENTED.

   Short Name  autoshare_revert
   Accessible By  A principal in full control of the record.

   URL Parameters
   • RECORD_ID – The id string associated with the Indivo record
• **CARENET_ID** – The id string associated with the Indivo carenet

• **DOCUMENT_ID** – The unique identifier of the Indivo document

**Returns** 200 Success.

Example Return Value:

<ok/>

---

**Todo**

The API Call ‘POST /records/{0}/documents/{1}/carenets/{2}/autoshare-revert’ is not yet implemented.

---

**PUT /records/(RECORD_ID)/documents/(DOCUMENT_ID)/label**

Set the label of a record-specific document.

**Short Name** record_document_label

**Accessible By** A user app with access to the record, or a principal in full control of the record

**URL Parameters**

• **RECORD_ID** – The id string associated with the Indivo record

• **DOCUMENT_ID** – The unique identifier of the Indivo document

**Raw Data** The new label for the document

**Returns** 200 OK with metadata describing the re-labeled document, or 404 Not Found if DOCUMENT_ID is invalid.

Example Return Value:

```
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
    <label>RELABELED: New HBA1C reading</label>
    <status>active</status>
    <nevershare>false</nevershare>
    <relatesTo>
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
      <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
    </relatesTo>
    <isRelatedFrom>
      <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
    </isRelatedFrom>
  </latest>
</Document>
```

---

**GET /records/(RECORD_ID)/documents/(DOCUMENT_ID)/meta**

Fetch the metadata of a record-specific document.
Short Name  record_document_meta

Accessible By  A user app with access to the record, or a principal in full control of the record

URL Parameters

- RECORD_ID – The id string associated with the Indivo record
- DOCUMENT_ID – The unique identifier of the Indivo document

Returns  200 OK with the document metadata, or 404 Not Found if DOCUMENT_ID is invalid.

Example Return Value:

```
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" />
  <label>HBA1C reading</label>
  <status>active</status>
  <nevershare>false</nevershare>
  <relatesTo>
    <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
  </relatesTo>
  <isRelatedFrom>
    <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
  </isRelatedFrom>
</Document>
```

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/meta
Set metadata fields on a document. NOT IMPLEMENTED.

Short Name  update_document_meta

Accessible By  Nobody

URL Parameters

- RECORD_ID – The id string associated with the Indivo record
- DOCUMENT_ID – The unique identifier of the Indivo document

Returns  200 Success.

Example Return Value:

```
<ok/>
```

Todo
The API Call ‘PUT /records/{0}/documents/{1}/meta’ is not yet implemented.
DELETE /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare
Remove the nevershare flag from a document.

**Short Name**  document_remove_nevershare

**Accessible By**  A principal in full control of the record.

**URL Parameters**
- **RECORD_ID** – The id string associated with the Indivo record
- **DOCUMENT_ID** – The unique identifier of the Indivo document

**Returns** 200 Success, or 404 Not Found if DOCUMENT_ID is invalid.

Example Return Value:

```
<ok/>
```

---

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare
Flag a document to never be shared, anywhere.

**Short Name**  document_set_nevershare

**Accessible By**  A principal in full control of the record.

**URL Parameters**
- **RECORD_ID** – The id string associated with the Indivo record
- **DOCUMENT_ID** – The unique identifier of the Indivo document

**Returns** 200 Success, or 404 Not Found if DOCUMENT_ID is invalid.

Example Return Value:

```
<ok/>
```

---

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL}/
Get all documents related to the passed document_id by a relation of the passed relation-type.

**Short Name**  get_documents_by_rel

**Accessible By**  A user app with access to the record, or a principal in full control of the record

**URL Parameters**
- **RECORD_ID** – The id string associated with the Indivo record
- **REL** – The type of relationship between the documents, i.e. annotation, interpretation
- **DOCUMENT_ID** – The unique identifier of the Indivo document

**Query Parameters**
- **status** – The account or document status to filter by.
- **limit** – See Query Operators. CURRENTLY UNIMPLEMENTED.
- **order_by** – See Query Operators. CURRENTLY UNIMPLEMENTED.
- **offset** – See Query Operators. CURRENTLY UNIMPLEMENTED

**Returns** 200 OK with a list of related documents, or 400 Bad Request if DOCUMENT_ID is invalid.
Example Return Value:

```xml
<Documents record_id="123" total_document_count="4">
  <Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98c6d9d5" size="77">
    <createdAt>2009-05-04T17:05:33</createdAt>
    <creator id="steve@indivo.org" type="account">
      <fullname>Steve Zabak</fullname>
    </creator>
    <suppressAt>2009-05-06T17:05:33</suppressAt>
    <suppressor id="steve@indivo.org" type="account">
      <fullname>Steve Zabak</fullname>
    </suppressor>
    <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
    <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" />
    <label>HBA1C reading</label>
    <status>active</status>
    <nevershare>false</nevershare>
    <relatesTo>
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
      <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
    </relatesTo>
    <isRelatedFrom>
      <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    </isRelatedFrom>
  </Document>
...</Documents>
```

**POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL}/**

Create a document and relate it to an existing document.

**Short Name**  document_create_by_rel

**Accessible By**  A user app with access to the record, or a principal in full control of the record

**URL Parameters**

- **RECORD_ID**  – The id string associated with the Indivo record
- **REL**  – The type of relationship between the documents, i.e. annotation, interpretation
- **DOCUMENT_ID**  – The unique identifier of the Indivo document

**Raw Data**  The raw content of the document to create.

**Returns**  200 OK with the metadata of the created document, or 400 Bad Request if the new content was invalid, or 404 Not Found if DOCUMENT_ID or REL are invalid.

Example Return Value:

```xml
<Document id="238543a5-e516-4da2-8a70-8c764c65a5a7" type="" size="104" digest="6aaef7aa0fbc24eef18f6f65a73b36" />
  <createdAt>2012-07-10T14:58:36.805563Z</createdAt>
  <creator id="bob@indivo.org" type="account">
    <fullname>Bob Loblaw</fullname>
  </creator>
  <original id="238543a5-e516-4da2-8a70-8c764c65a5a7" />
  <status>active</status>
  <nevershare>false</nevershare>
</Document>
```
POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL}/external/{PHA_EMAIL}/{EXTERNAL_ID}

Create a document, assign it an external id, and relate it to an existing document.

**Short Name**  
document_create_by_rel_with_ext_id

**Accessible By**  
A user app with access to the record, with an id matching the app email in the URL.

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record
- **REL** – The type of relationship between the documents, i.e. annotation, interpretation
- **EXTERNAL_ID** – The external identifier of the desired resource
- **PHA_EMAIL** – The email identifier of the Indivo user app
- **DOCUMENT_ID** – The unique identifier of the Indivo document

**Raw Data**  
The raw content of the document to create.

**Returns**  
200 OK with the metadata of the created document, or 400 Bad Request if the new content was invalid, or 404 Not Found if DOCUMENT_ID or REL are invalid.

Example Return Value:

```xml
<Document id="238543a5-e516-4da2-8a70-8c764c65a5a7" type="" size="104" digest="6aaef7aa0fbc24eef18f6d6843f12d762f3b915" createdAt="2012-07-10T14:58:36.805563Z" creator="bob@indivo.org" type="account">  
  <fullName>Bob Loblaw</fullName>  
  <createdAt>2012-07-10T14:58:36.805563Z</createdAt>  
  <creator id="bob@indivo.org" type="account">  
    <fullname>Bob Loblaw</fullname>  
  </creator>  
  <original id="238543a5-e516-4da2-8a70-8c764c65a5a7"/>  
  <status>active</status>  
  <nevershare>false</nevershare>  
  <isRelatedFrom>  
    <relation type="http://indivo.org/vocab/documentrels#annotation" count="1" />  
  </isRelatedFrom>  
</Document>
```

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL}/external/{PHA_EMAIL}/{EXTERNAL_ID}

Create a document, assign it an external id, and relate it to an existing document.

**Short Name**  
document_create_by_rel_with_ext_id

**Accessible By**  
A user app with access to the record, with an id matching the app email in the URL.

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record
- **REL** – The type of relationship between the documents, i.e. annotation, interpretation
- **EXTERNAL_ID** – The external identifier of the desired resource
- **PHA_EMAIL** – The email identifier of the Indivo user app

**Example Return Value:**
• DOCUMENT_ID – The unique identifier of the Indivo document

Raw Data  The raw content of the document to create.

Returns 200 Success, 400 Bad Request if the new content was invalid, or 404 Not Found if DOCUMENT_ID or REL are invalid.

Example Return Value:

<pre><code>&lt;ok/&gt;</code></pre>

**POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace**

Create a new version of a record-specific document.

Short Name  document_version

Accessible By  A user app with access to the record, a principal in full control of the record, or the admin app that created the record.

URL Parameters

• RECORD_ID – The id string associated with the Indivo record  
  • DOCUMENT_ID – The unique identifier of the Indivo document

Raw Data  The raw content of the document to create.

Returns 200 OK with metadata on the new document, 400 Bad Request if the old document has already been replaced by a newer version, or 404 Not Found if DOCUMENT_ID is invalid or if the new content is invalid.

Example Return Value:

<pre><code>&lt;Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">  &lt;createdAt&gt;2009-05-04T17:05:33&lt;/createdAt&gt;  &lt;creator id="steve@indivo.org" type="account"> &lt;fullname&gt;Steve Zabak&lt;/fullname&gt; &lt;/creator&gt;  &lt;suppressedAt&gt;2009-05-06T17:05:33&lt;/suppressedAt&gt;  &lt;suppressor id="steve@indivo.org" type="account"> &lt;fullname&gt;Steve Zabak&lt;/fullname&gt; &lt;/suppressor&gt;  &lt;replaces id="abe8130e2-ba54-1234-eeef-45a3b6cd9a8e" /&gt;  &lt;original id="abe8130e2-ba54-1234-eeef-45a3b6cd9a8e" /&gt;  &lt;latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org" label="HBA1C reading"&gt; &lt;status&gt;active&lt;/status&gt; &lt;nevershare&gt;false&lt;/nevershare&gt;  &lt;relatesTo&gt; &lt;relation type="http://indivo.org/vocab/documentrels#attachment" count="1" /&gt; &lt;/relatesTo&gt; &lt;isRelatedFrom&gt; &lt;relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" /&gt; &lt;/isRelatedFrom&gt; &lt;/Document&gt;
</code></pre>

**PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace/external/{PHA_EMAIL}/{EXTERNAL_ID}**

Create a new version of a record-specific document and assign it an external id.
**Short Name** document_version_by_ext_id

**Accessible By** A user app with access to the record, with an id matching the app email in the URL.

**URL Parameters**
- RECORD_ID – The id string associated with the Indivo record
- EXTERNAL_ID – The external identifier of the desired resource
- PHA_EMAIL – The email identifier of the Indivo user app
- DOCUMENT_ID – The unique identifier of the Indivo document

**Raw Data** The raw content of the document to create.

**Returns** 200 OK with metadata on the new document, 400 Bad Request if the old document has already been replaced by a newer version, or 404 Not Found if DOCUMENT_ID is invalid or if the new content is invalid.

Example Return Value:

```xml
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d546e4d8e5f89f993a797da39f0c8" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <replaces id="abe8130e2-ba54-1234-eeef-45a3b6cd9a8e" />
  <original id="abe8130e2-ba54-1234-eeef-45a3b6cd9a8e" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org"/>
  <label>HBA1C reading</label>
  <status>active</status>
  <nevershare>false</nevershare>
  <relatesTo>
    <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
  </relatesTo>
  <isRelatedFrom>
    <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
  </isRelatedFrom>
</Document>
```

**POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/set-status**

Set the status of a record-specific document.

**Short Name** document_set_status

**Accessible By** A user app with access to the record, or a principal in full control of the record

**URL Parameters**
- RECORD_ID – The id string associated with the Indivo record
- DOCUMENT_ID – The unique identifier of the Indivo document

**FormData Parameters**
- status – The new status for the document. Options are active, void, archived.
- reason – The reason for the status change.
Returns 200 Success, 400 Bad Request if `status` or `reason` are missing, or 404 Not Found if `DOCUMENT_ID` is invalid.

Example Return Value:

```
<ok/>
```

---

GET `/records/{RECORD_ID}/documents/{DOCUMENT_ID}/status-history`
List all changes to a document’s status over time.

Short Name  document_status_history
Accessible By  A user app with access to the record, or a principal in full control of the record

URL Parameters
- `RECORD_ID` – The id string associated with the Indivo record
- `DOCUMENT_ID` – The unique identifier of the Indivo document

Returns 200 OK with a the document’s status history, or 404 Not Found if `DOCUMENT_ID` is invalid.

Example Return Value:

```
<DocumentStatusHistory document_id="456">
  <DocumentStatus by="joeuser@indivo.example.org" at="2010-09-03T12:45:12Z" status="archived">
    <reason>no longer relevant</reason>
  </DocumentStatus>
  ...
</DocumentStatusHistory>
```

---

GET `/records/{RECORD_ID}/documents/{DOCUMENT_ID}/versions/`
Retrieve the versions of a document.

Short Name  document_versions
Accessible By  A user app with access to the record, or a principal in full control of the record

URL Parameters
- `RECORD_ID` – The id string associated with the Indivo record
- `DOCUMENT_ID` – The unique identifier of the Indivo document

Query Parameters
- `status` – The account or document status to filter by.
- `limit` – See Query Operators.
- `order_by` – See Query Operators.
- `offset` – See Query Operators.

Returns 200 OK with a list of document versions, or 404 Not Found if `DOCUMENT_ID` is invalid.

Example Return Value:
POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}

Send a message to a record.

**Short Name** record_send_message

**Accessible By** Any admin app, or a user app with access to the record.

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record
- **MESSAGE_ID** – The unique external identifier of the Indivo Message, for idempotency.

**FormData Parameters**

- **body** – The message body. Defaults to [no body].
- **body_type** – The formatting for the message body. Options are plaintext, markdown. Defaults to plaintext.
- **num_attachments** – The number of attachments this message requires. Attachments are uploaded with calls to POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}. Defaults to 0.
- **severity** – The importance of the message. Options are low, medium, high. Defaults to low.
- **subject** – The message subject. Defaults to [no subject].

**Returns** 200 Success, or 400 Bad Request if MESSAGE_ID was a duplicate. Also triggers notification emails to accounts authorized to view messages for the passed record.
Example Return Value:

<ok/>

POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}
Attach a document to an Indivo message.

Short Name record_message_attach
Accessible By Any admin app, or a user app with access to the record.

URL Parameters

• RECORD_ID – The id string associated with the Indivo record
• ATTACHMENT_NUM – The 1-indexed number corresponding to the message attachment
• MESSAGE_ID – The unique identifier of the Indivo Message

Raw Data The raw XML attachment data.
Returns 200 Success, or 400 Bad Request if ATTACHMENT_NUM has already been uploaded.

Example Return Value:

<ok/>

POST /records/{RECORD_ID}/notifications/
Send a notification about a record to all accounts authorized to be notified.

Short Name record_notify
Accessible By Any admin app, or a user app with access to the record.

URL Parameters

• RECORD_ID – The id string associated with the Indivo record

FormData Parameters

• content – The plaintext content of the notification.
• app_url – A callback url to the app for more information. OPTIONAL.
• document_id – The id of the document to which this notification pertains. OPTIONAL.

Returns 200 Success, or 400 Bad Request if content wasn’t passed.

Example Return Value:

<ok/>

POST /records/{RECORD_ID}/notify
Send a notification about a record to all accounts authorized to be notified.

Short Name record_notify
Accessible By Any admin app, or a user app with access to the record.

URL Parameters

• RECORD_ID – The id string associated with the Indivo record
Formdata Parameters

- **content** – The plaintext content of the notification.
- **app_url** – A callback url to the app for more information. **OPTIONAL.**
- **document_id** – The id of the document to which this notification pertains. **OPTIONAL.**

**Returns** 200 Success, or 400 Bad Request if `content` wasn’t passed.

Example Return Value:

```
<ok/>
```

Deprecated since version 1.0: Use `POST /records/{RECORD_ID}/notifications/` instead.

---

**GET /records/{RECORD_ID}/owner**

Get the owner of a record.

**Short Name** record_get_owner

**Accessible By** A principal in full control of the record, or any admin app.

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record

**Returns** 200 Success.

Example Return Value:

```
<Account id='joeuser@example.com' />
```

---

**POST /records/{RECORD_ID}/owner**

Set the owner of a record.

**Short Name** record_set_owner

**Accessible By** Any admin app.

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record

**Raw Data** The email address of the new account owner.

**Returns** 200 OK with information about the account, or 400 Bad Request if the passed email address is invalid.

Example Return Value:

```
<Account id="joeuser@indivo.example.org">
  <fullName>Joe User</fullName>
  <contactEmail>joeuser@gmail.com</contactEmail>
  <lastLoginAt>2010-05-04T15:34:23Z</lastLoginAt>
  <totalLoginCount>43</totalLoginCount>
  <failedLoginCount>0</failedLoginCount>
  <state>active</state>
  <lastStateChange>2009-04-03T13:12:12Z</lastStateChange>

  <authSystem name="password" username="joeuser" />
  <authSystem name="hospital_sso" username="Joe_User" />
</Account>
```
**PUT /records/{RECORD_ID}/owner**
Set the owner of a record.

**Short Name** record_set_owner

**Accessible By** Any admin app.

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record

**Raw Data** The email address of the new account owner.

**Returns** 200 OK with information about the account, or 400 Bad Request if the passed email address is invalid.

Example Return Value:

```xml
<Account id="joeuser@indivo.example.org">
  <fullName>Joe User</fullName>
  <contactEmail>joeuser@gmail.com</contactEmail>
  <lastLoginAt>2010-05-04T15:34:23Z</lastLoginAt>
  <totalLoginCount>43</totalLoginCount>
  <failedLoginCount>0</failedLoginCount>
  <state>active</state>
  <lastStateChange>2009-04-03T13:12:12Z</lastStateChange>
  <authSystem name="password" username="joeuser" />
  <authSystem name="hospital_sso" username="Joe_User" />
</Account>
```

---

**GET /records/{RECORD_ID}/reports/experimental/ccr**
Export patient data as a Continuity of Care Record (CCR) document.

**Short Name** report_ccr

**Accessible By** A user app with access to the record, or a principal in full control of the record

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record

**Returns** 200 OK with an EXPERIMENTAL CCR document.

Example Return Value:

```xml
<ContinuityOfCareRecord xmlns="urn:astm-org:CCR">
  <CCRDocumentObjectID>0</CCRDocumentObjectID>
  <Language>
    <Text>ENGLISH</Text>
  </Language>
  <Version>V1.0</Version>
  <DateTime>
    <Type>Create</Type>
    <ObjectAttribute>
      <Attribute>DisplayDate</Attribute>
      <AttributeValue>
        <Value>09/30/10</Value>
      </AttributeValue>
    </ObjectAttribute>
  </DateTime>
</ContinuityOfCareRecord>
```
<Patient>
  <ActorID>123</ActorID>
</Patient>

<From>
  <ActorLink/>
</From>

<Body>
  <Medications>
    <Medication>
      <CCRDataObjectID>789</CCRDataObjectID>
      <DateTime>
        <Type>
          <Text>Dispense date</Text>
        </Type>
        <ExactDateTime>2010-05-04T15:34:23Z</ExactDateTime>
      </DateTime>
      <Status>
        <Text>Active</Text>
      </Status>
      <Product>
        <ProductName>
          <Text>Vioxx</Text>
        </ProductName>
        <Strength>
          <Value>20</Value>
          <Units>
            <Unit>mg</Unit>
          </Units>
        </Strength>
        <Directions>
          <Dose>
            <Value>1</Value>
            <Units>
              <Unit>Pills</Unit>
            </Units>
          </Dose>
          <Route>
            <Text>Oral</Text>
          </Route>
          <Frequency>
            <Value>1QR</Value>
          </Frequency>
        </Directions>
      </Product>
    </Medication>
  </Medications>
</Body>

...
GET /records/{RECORD_ID}/reports/minimal/equipment/
List the equipment data for a given record.

**Short Name** equipment_list

**Accessible By** A user app with access to the record, or a principal in full control of the record

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record

**Query Parameters**

- **status** – The account or document status to filter by
- **{FIELD}** – See Query Operators, Valid Query Fields
- **date_group** – See Query Operators
- **group_by** – See Query Operators
- **order_by** – See Query Operators
- **aggregate_by** – See Query Operators

3.3. API Reference 189
Indivo X Documentation, Release 2.0.0

- **date_range** – See Query Operators
- **limit** – See Query Operators
- **offset** – See Query Operators

**Returns** 200 OK with a list of equipment, or 400 Bad Request if any invalid query parameters were passed.

**Example Return Value:**

```
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*"/>
    <Filters>
      <Filter name="equipment_name" value="pacemaker"/>
    </Filters>
  </QueryParams>
  <Report>
    <Meta>
      <Document id="261ca370-927f-41af-b001-7b615c7a468e" type="http://indivo.org/vocab/xml/documents#Models" size="1653" digest="0799971784e5a2d199cd6585415a8cd57f7bf9e4f8c8f74ef67a1009a1481cd6" record_id="">
        <createdAt>2011-05-02T17:48:13Z</createdAt>
        <creator id="mymail@mail.ma" type="Account">
          <fullname>full name</fullname>
        </creator>
        <original id="261ca370-927f-41af-b001-7b615c7a468e"/>
        <label>testing</label>
        <status>active</status>
        <nevershare>false</nevershare>
      </Document>
    </Meta>
    <Item>
      <Equipment xmlns="http://indivo.org/vocab/xml/documents#">
        <dateStarted>2009-02-05</dateStarted>
        <dateStopped>2010-06-12</dateStopped>
        <type>cardiac</type>
        <name>Pacemaker</name>
        <vendor>Acme Medical Devices</vendor>
        <id>167-ABC-23</id>
        <description>it works</description>
        <specification>blah blah blah</specification>
      </Equipment>
    </Item>
  </Report>
  ...
</Reports>
```

**GET /records/{RECORD_ID}/reports/minimal/measurements/{LAB_CODE}/**

List the measurement data for a given record.

**Short Name** measurement_list

**Accessible By** A user app with access to the record, or a principal in full control of the record

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record
• **LAB_CODE** – The identifier corresponding to the measurement being made.

**Query Parameters**

• **status** – The account or document status to filter by

• **{FIELD}** – See Query Operators, Valid Query Fields

• **date_group** – See Query Operators

• **group_by** – See Query Operators

• **order_by** – See Query Operators

• **aggregate_by** – See Query Operators

• **date_range** – See Query Operators

• **limit** – See Query Operators

• **offset** – See Query Operators

**Returns** 200 OK with a list of measurements, or 400 Bad Request if any invalid query parameters were passed.

**Example Return Value:**

```xml
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*" />
    <Filters>
      <Filter name="lab_type" value="hematology"/>
    </Filters>
  </QueryParams>
  <Report>
    <Meta>
      <Document id="261ca370-927f-41af-b001-7b615c7a468e" type="http://indivo.org/vocab/xml/documents#Measurement" size="1653" digest="0799971784e5a2d199cd6585415a8cd57f7bf9e4f8c8f74ef67a1009a1481cd6" record_id="">
        <createdAt>2011-05-02T17:48:13Z</createdAt>
        <creator id="mymail@mail.ma" type="Account">
          <fullname>full name</fullname>
        </creator>
        <original id="261ca370-927f-41af-b001-7b615c7a468e"/>
        <label>testing</label>
        <status>active</status>
        <nevershare>false</nevershare>
      </Document>
    </Meta>
    <Item>
      <Measurement id="1234" value="120" type="blood pressure systolic" datetime="2011-03-02T00:00:00Z" unit="mmHg" source_doc="3456" />
    </Item>
  </Report>
  ...
</Reports>
```

**GET /records/{RECORD_ID}/reports/minimal/procedures/**

List the procedure data for a given record.

**Short Name** procedure_list

**Accessible By** A user app with access to the record, or a principal in full control of the record

3.3. API Reference 191
URL Parameters

- RECORD_ID – The id string associated with the Indivo record

Query Parameters

- status – The account or document status to filter by
- [FIELD] – See Query Operators, Valid Query Fields
- date_group – See Query Operators
- group_by – See Query Operators
- order_by – See Query Operators
- aggregate_by – See Query Operators
- date_range – See Query Operators
- limit – See Query Operators
- offset – See Query Operators

Returns 200 OK with a list of procedures, or 400 Bad Request if any invalid query parameters were passed.

Example Return Value:

```xml
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*" />
    <Filters>
      </Filters>
  </QueryParams>
  <Report>
    <Meta>
      <Document id="261ca370-927f-41af-b001-7b615c7a468e" type="http://indivo.org/vocab/xml/documents#Procedure">
        <createdAt>2011-05-02T17:48:13Z</createdAt>
        <creator id="mymail@mail.ma" type="Account">
          <fullname>full name</fullname>
        </creator>
        <original id="261ca370-927f-41af-b001-7b615c7a468e" />
        <label>testing</label>
        <status>active</status>
        <nevershare>false</nevershare>
      </Document>
    </Meta>
    <Item>
      <Procedure xmlns="http://indivo.org/vocab/xml/documents#">
        <datePerformed>2009-05-16T12:00:00</datePerformed>
        <name type="http://codes.indivo.org/procedures#" value="85" abbrev="append">Appendectomy</name>
        <provider>
          <name>Kenneth Mandl</name>
          <institution>Children’s Hospital Boston</institution>
        </provider>
      </Procedure>
    </Item>
  </Report>
</Reports>
```
GET /records/{RECORD_ID}/reports/minimal/simple-clinical-notes/
List the simple_clinical_notes data for a given record.

**Short Name**  simple_clinical_notes_list  

**Accessible By**  A user app with access to the record, or a principal in full control of the record

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record

**Query Parameters**

- **status** – The account or document status to filter by
- **{FIELD}** – See Query Operators, Valid Query Fields
- **date_group** – See Query Operators
- **group_by** – See Query Operators
- **order_by** – See Query Operators
- **aggregate_by** – See Query Operators
- **date_range** – See Query Operators
- **limit** – See Query Operators
- **offset** – See Query Operators

**Returns**  200 OK with a list of notes, or 400 Bad Request if any invalid query parameters were passed.

Example Return Value:

```xml
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*" />
    <Filters>
    </Filters>
  </QueryParams>
  <Report>
    <Meta>
      <Document id="261ca370-927f-41af-b001-7b615c7a468e" type="http://indivo.org/vocab/xml/documents#">
        <createdAt>2011-05-02T17:48:13Z</createdAt>
        <creator id="mymail@mail.ma" type="Account">
          <fullname>full name</fullname>
        </creator>
        <original id="261ca370-927f-41af-b001-7b615c7a468e"/>
        <label>testing</label>
        <status>active</status>
        <nevershare>false</nevershare>
      </Document>
    </Meta>
    <Item>
      <SimpleClinicalNote xmlns="http://indivo.org/vocab/xml/documents#">
        <dateOfVisit>2010-02-02T12:00:00Z</dateOfVisit>
        <finalizedAt>2010-02-03T13:12:00Z</finalizedAt>
      </SimpleClinicalNote>
    </Item>
  </Report>
</Reports>
```
<visitType type="http://codes.indivo.org/visit-types#" value="acute">Acute Care</visitType>
<visitLocation>Longfellow Medical</visitLocation>
<specialty type="http://codes.indivo.org/specialties#" value="hem-onc">Hematology/Oncology</specialty>

<signature>
  <at>2010-02-03T13:12:00Z</at>
  <provider>
    <name>Kenneth Mandl</name>
    <institution>Children’s Hospital Boston</institution>
  </provider>
</signature>

<signature>
  <provider>
    <name>Isaac Kohane</name>
    <institution>Children’s Hospital Boston</institution>
  </provider>
</signature>

<chiefComplaint>stomach ache</chiefComplaint>
<content>Patient presents with ...</content>
</SimpleClinicalNote>
</Item>
</Report>

...
Returns 200 OK with a list of DATA_MODELs, or 400 Bad Request if any invalid query parameters were passed.

Example Return Value:

SDMJ Example:

```
{
    "__modelname__": "TestMedication",
    "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
    "name": "ibuprofen",
    "date_started": "2010-10-01T00:00:00Z",
    "date_stopped": "2010-10-31T00:00:00Z",
    "brand_name": "Advil",
    "prescription": {
        "__modelname__": "TestPrescription",
        "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
        "prescribed_by_name": "Kenneth D. Mandl",
        "prescribed_by_institution": "Children’s Hospital Boston",
        "prescribed_on": "2010-09-30T00:00:00Z",
        "prescribed_stop_on": "2010-10-31T00:00:00Z"
    },
    "fills": [
        {
            "__modelname__": "TestFill",
            "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
            "date_filled": "2010-10-01T00:00:00Z",
            "supply_days": "15",
            "filled_at_name": "CVS"
        },
        {
            "__modelname__": "TestFill",
            "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
            "date_filled": "2010-10-16T00:00:00Z",
            "supply_days": "15",
            "filled_at_name": "CVS"
        }
    ]
}
```

SDMX Example:

```
<Models>
    <Model name="TestMedication" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
        <Field name="date_started">2010-10-01T00:00:00Z</Field>
        <Field name="name">ibuprofen</Field>
        <Field name="brand_name">Advil</Field>
        <Field name="date_stopped">2010-10-31T00:00:00Z</Field>
        <Field name="prescription">
            <Model name="TestPrescription" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
                <Field name="prescribed_by_name">Kenneth D. Mandl</Field>
                <Field name="prescribed_by_institution">Children’s Hospital Boston</Field>
                <Field name="prescribed_on">2010-09-30T00:00:00Z</Field>
                <Field name="prescribed_stop_on">2010-10-31T00:00:00Z</Field>
            </Model>
        </Field>
        <Field name="fills">
            <Models>
                <Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
                </Model>
            </Models>
        </Field>
    </Model>
</Models>
```
GET /records/{RECORD_ID}/shares/
List the shares of a record.

Short Name record_shares
Accessible By The owner of the record, or any admin app.

URL Parameters

• RECORD_ID – The id string associated with the Indivo record

Returns 200 OK with a list of shares.

Example Return Value:

<Shares record="123">
  <Share id="678" account="joeuser@example.com" />
  <Share id="789" pha="problems@apps.indivo.org" />

  ...
</Shares>

POST /records/{RECORD_ID}/shares/
Fully share a record with another account.

Short Name record_share_add
Accessible By The owner of the record, or any admin app.

URL Parameters

• RECORD_ID – The id string associated with the Indivo record

Formdata Parameters

• account_id – The email address of the recipient account. REQUIRED.

• role_label – A label for the share, usually the relationship between the owner and the recipient (i.e. Guardian). OPTIONAL.

Returns 200 Success, 400 Bad Request if account_id was not passed, or 404 Not Found if the passed account_id was invalid.

Example Return Value:
DELETE /records/{RECORD_ID}/shares/{OTHER_ACCOUNT_ID}
Undo a full record share with an account.

Short Name record_share_delete
Accessible By The owner of the record, or any admin app.

URL Parameters
• RECORD_ID – The id string associated with the Indivo record
• OTHER_ACCOUNT_ID – The email identifier of the Indivo account to share with

Returns 200 Success, or 404 Not Found if OTHER_ACCOUNT_ID is invalid.

Example Return Value:
<ok/>

POST /records/{RECORD_ID}/shares/{OTHER_ACCOUNT_ID}/delete
Undo a full record share with an account.

Short Name record_share_delete
Accessible By The owner of the record, or any admin app.

URL Parameters
• RECORD_ID – The id string associated with the Indivo record
• OTHER_ACCOUNT_ID – The email identifier of the Indivo account to share with

Returns 200 Success, or 404 Not Found if OTHER_ACCOUNT_ID is invalid.

Example Return Value:
<ok/>

Deprecated since version 1.0: Use DELETE /records/{RECORD_ID}/shares/{OTHER_ACCOUNT_ID} instead.

GET /records/{RECORD_ID}/{MODEL_NAME}/
SMART-compatible alias for the generic list view: returns data_models serialized as SMART RDF.

Short Name smart_generic
Accessible By A user app with access to the record, or a principal in full control of the record

URL Parameters
• RECORD_ID – The id string associated with the Indivo record
• MODEL_NAME – The name of the SMART data_model to retrieve (i.e. problems). Options are defined by the SMART API.

Returns 200 OK with SMART RDF/XML for all items matching MODEL_NAME belonging to the record.

Example Return Value:
New in version 2.0.0.

GET /records/{RECORD_ID}/{MODEL_NAME}/{MODEL_ID}

Retrieve a specific instance of a SMART model.

**Short Name** smart_generic_instance

**Accessible By** A user app with access to the record, or a principal in full control of the record

**URL Parameters**

- **RECORD_ID** – The id string associated with the Indivo record
- **MODEL_ID** – The id of the SMART data_model to retrieve
- **MODEL_NAME** – The name of the SMART data model to retrieve (i.e. problems). Options are defined by the SMART API.

**Returns** 200 OK with SMART RDF/XML for the item matching **MODEL_NAME** and **MODEL_ID** belonging to the record.

Example Return Value:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:sp="http://smartplatforms.org/terms#"
  xmlns:v="http://www.w3.org/2006/vcard/ns#">
  <rdf:Description rdf:about="http://smartplatforms.org/terms/codes/LabResultInterpretation#normal">
    <sp:system>http://smartplatforms.org/terms/codes/LabResultInterpretation#</sp:system>
  </rdf:Description>
</rdf:RDF>
```
Normal

Serum Sodium

final results: complete and verified

Normal

City Lab
<rdf:Description rdf:nodeID="_d8aacfc9-e1f0-4702-8f8e-c1846c5c214a">
    <sp:unit>mEq/L</sp:unit>
    <rdf:type rdf:resource="http://smartplatforms.org/terms#ValueAndUnit"/>
    <sp:value>145</sp:value>
</rdf:Description>

<rdf:Description rdf:about="http://purl.bioontology.org/ontology/LNC/2951-2">
    <sp:system>http://purl.bioontology.org/ontology/LNC/</sp:system>
    <rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
    <dcterms:title>Serum Sodium</dcterms:title>
    <dcterms:identifier>2951-2</dcterms:identifier>
</rdf:Description>

<rdf:Description rdf:about="http://indivo.org/records/f48030f2-5ed6-4fc0-97b2-a925d7d88d94/lab_results/d2ed9b5e-3eab-4cae-a35c-f5d8ce4c9fff">
    <sp:abnormalInterpretation rdf:nodeID="_c2141794-697e-4917-8cb2-eefc7d612f1a"/>
    <sp:notes>Blood sample appears to have hemolyzed</sp:notes>
    <sp:belongsTo rdf:resource="http://indivo.org/records/f48030f2-5ed6-4fc0-97b2-a925d7d88d94"/>
    <rdf:type rdf:resource="http://smartplatforms.org/terms#LabResult"/>
    <sp:quantitativeResult rdf:nodeID="_6fa51a50-1bdc-43b9-8be3-cbe828f2hab220"/>
    <sp:labName rdf:nodeID="_dc97d85d-c1fd-4996-980a-f2b37195c00a"/>
    <sp:accessionNumber>AC09205823577</sp:accessionNumber>
    <sp:labStatus rdf:nodeID="_7e185aa4-7394-4ce8-b107-d3ae0f1318c"/>
    <sp:specimenCollected rdf:nodeID="_0e2687ff-7b32-430e-b04e-d21f3349184e"/>
</rdf:Description>

<rdf:Description rdf:about="http://smartplatforms.org/terms/codes/LabStatus#final">
    <sp:system>http://smartplatforms.org/terms/codes/LabStatus#</sp:system>
    <rdf:type rdf:resource="http://smartplatforms.org/terms#Code"/>
    <dcterms:title>Final results: complete and verified</dcterms:title>
    <dcterms:identifier>final</dcterms:identifier>
</rdf:Description>

<rdf:Description rdf:nodeID="_6e0921df-5c19-4a7f-b0c3-aca46ff8d2"/>

<rdf:Description rdf:nodeID="_0ac2cdea-f2b3-44c6-b703-284a757111d8">
    <v:family-name>Finnialispi</v:family-name>
    <v:given-name>Tad</v:given-name>
</rdf:Description>
3.4 App Size Conventions

If your app is aimed at the general web user population, your app should render correctly in a <DIV> that is maximum 768 pixels wide!

Indivo UI assumes that the user’s minimum browser size is 1024 pixels wide by 768 pixels high. Taking Indivo UI’s interface elements into account, this leaves 768 pixels of width for your app content.

The safe assumption is that your app content has 768px of width, but no more

Many users will have screen resolutions that make this the maximum width of app content they will be able to display. If your app exceeds 768 pixels wide by default, the majority of the general web user’s browsers will side-scroll, which is poor user experience and to be avoided. For users with higher resolution screens, the Indivo UI will expand and provide your app with more pixels of width that you can take advantage of using a “liquid” layout in your app, but a wider screen cannot be assumed by default for the general web user population.

As for length, there are less strict criteria, since the length of the app_content_pane will expand to fit longer content and we assume that users will scroll vertically. Be aware that empirical browser size research shows that for 90% of users the bottom edge of their browser window is at approximately 500 pixels and for 80% of users the bottom edge is around 560 pixels. To provide a good experience for your users it’s important to keep key data and interface elements “above the fold” e.g. above 500 to 560 pixels.

3.5 Indivo Authentication

As of version 2.0, Indivo now permits three methods of authentication:

- Traditional oAuth
- In-browser Connect Authentication (a la. SMART CONNECT)
- Pre-generated REST Authentication (a la. SMART REST)
3.5.1 Indivo oAuth

Here, we provide details of exactly how Indivo uses OAuth, in particular the options that are supported and those that aren’t.

Specifics of our OAuth implementation

OAuth Section 5.2 defines three possible approaches to sending OAuth Protocol Parameters. In Indivo, we use exclusively the HTTP authorization header, as defined by the OAuth specification, as the preferred method. We allow oauth_callback and oauth_verifier to be provided as POST parameters, only because some client libraries that are not OAuth 1.0a-compatible cannot otherwise connect.

We require all changes suggested by the OAuth 1.0a revision:

- When asking for a request_token, the PHA must provide oauth_callback as an extra OAuth parameter. This can be a URL where Indivo X will send the user after token authorization, or it can be oob to use the default URL specified by the PHA at registration time.
- When responding to a request token, Indivo X will include oauth_callback_confirmed=true
- A PHA’s callback URL should accept both oauth_token and oauth_verifier parameters.
- When exchanging a request token for an access token, the PHA must provide the corresponding oauth_verifier parameter in addition to all existing parameters.

In addition, we implement the following constraints:

- As per the OAuth recommended approach, all OAuth token setup and exchange calls use the POST method. The Indivo server will respond with a 405 Method Not Allowed error code to any GET request against its OAuth protocol URLs.
- The oauth_version parameter is mandatory. Every PHA request should include the oauth_version parameter. The only supported value in Indivo at this point is oauth_version=1.0.
- Given an Indivo Server running at https://INDIVO_SERVER, the OAuth URLs are defined as follows:
  - Request Token URL: https://INDIVO_SERVER/oauth/request_token
  - User Authorization URL: Since there is a UI component to enabling User Authorization (i.e., we have to obtain their consent), Indivo_Server does not explicitly offer a User Authorization URL via its API. Individual UI apps can (with a valid user session) authorize tokens on behalf of users with a call to POST /oauth/internal/request_tokens/{TOKEN}/approve, but each individual UI-app implementation will provide a different app-facing User Authorization URL. We recommend using https://UI_SERVER/oauth/authorize?oauth_token={token}, which is the URL used by our reference UI-app implementation.
  - Access Token URL: https://INDIVO_SERVER/oauth/access_token
- OAuth defines a number of default signature methods and leaves open the possibility of using other signature methods. Indivo supports only one request signature method: HMAC-SHA1. Support for RSA-SHA1 may eventually be added.

Body and Content-Type

By default, OAuth only signs the body of HTTP requests that are form-url-encoded. Indivo uses the OAuth Body Hash Extension to ensure that raw POSTs, e.g. to send XML documents, become part of the signature. When the body-hash extension is activated, Indivo also expects an additional parameter, oauth_content_type, to certify the content type HTTP header (and prevent content-sniffing attacks.)
User Applications (PHAs)

PHA Registration

A PHA registers with Indivo using an Indivo-installation specific process, at the conclusion of which both Indivo and the PHA agree on:

- The name of the PHA, e.g. “Medical Surveys”
- An OAuth consumer_key and consumer_secret.
- A start_url_template for the PHA, e.g. http://acme.com/indivoapp?record_id={indivo_record_id}&document_id={document_id}
- A callback_url for the PHA, e.g. http://acme.com/success_after_indivo, which should expect to receive query parameters oauth_token and oauth_verifier.
- Whether the PHA has a web user interface (certain applications that synchronize data have no UI), and whether that PHA is frameable inside Indivo.
- Whether the PHA is autonomous or not, and if it is, why it wants that kind of access to the user’s personal health record.

IMPORTANTLY, the callback_url is the only URL that Indivo will return the user to after a successful PHA attachment. Indivo does not support a custom OAuth callback URL.

Autonomous Apps

An autonomous app is one that wants to access the user’s record while the user is not connected. PHAs that qualify include hospital data connectors, drug-interaction checkers, etc. There are very good reasons for PHAs to access a record while the user is not online, but we want to ensure that users understand the implications, and thus the Indivo authorization pathway looks different depending on whether an app is autonomous or not.

An app must choose to be autonomous at registration time. It must be autonomous for all users, or for none.

An autonomous app accesses the entire record by default, and the user must consent to this. This design choice is meant to prevent medical mistakes for automated apps that, for example, check for drug-drug interactions but may fail to notify the user if they have only partial data access. An autonomous app thus triggers the appropriate authorization screen that warns the user about the long-term, autonomous access, displays the app’s reason for requesting this type of access, and simply gives the user a yes/no choice.

Autonomous apps can, in some circumstances, have no user-interface. This might happen if, for example, a hospital connector application sits behind the hospital firewall and connects autonomously to the Indivo record to upload hospital data into the PCHR, but never lets the user connect directly to the app itself. There are two ways, currently, to authorize such an application. The first is via admin-based PHA setup, where an administrative app primes the Indivo record with this app. Alternatively, the UI may allow users to permission autonomous apps without an interface. In any case, these apps must declare their lack of UI at registration time, much like they declare their being autonomous or not. Only autonomous apps can choose to forgo a UI.

A non-autonomous app, on the other hand, is one that is meant to be used by whoever is logged in and has access to the record in question. Depending on which user has launched the app, the app’s permissions might differ. For example, when Alice uses the Problems App within her record, she should see ‘all’ of her problems. However, when Bob, her co-worker, uses the Problems App to view Alice’s record, he should see only those problems which Alice has chosen to let him see. Thus, a non-autonomous app exists purely to proxy a human user’s clicks and perform some visualization / data entry assistance functionality. Non-autonomous apps are thus constrained to a carenet at the time that the user clicks on the app name to launch it. When Bob launches the Problems App on Alice’s record, the Problems App receives an access token that is constrained to Alice’s “Work” carenet, and the app can only access the...
problems Alice has made available within her Work carenet. All access tokens for non-autonomous apps are valid only for the duration of a web session.

**Connecting a PHA to a Record**

A user opts to add a PHA to her Indivo record by enabling it in the UI. At any subsequent point, when the user attempts to use the PHA (i.e. by clicking on it in the UI), she is sent to the PHA’s `start URL` with the `indivo_record_id` filled in. The PHA may present informational content if it so desires, then is expected to begin the OAuth authorization process. When the PHA begins the OAuth process, it should do so with the indicated `indivo_record_id` that it received when its `start_url` was accessed.

**Obtain a Request Token**

A PHA begins its access request for a user when the user visits the PHA’s `start URL`. While the user’s browser awaits a response, the PHA obtains from the Indivo Server a request token. This is accomplished by issuing a signed POST 2-legged OAuth request to the Request Token URL:

```
https://INDIVO_SERVER/oauth/request_token
```

with optional form parameter `indivo_record_id`. Again, if the PHA was accessed via its `start_url` with the Indivo record ID filled in, it should use this record ID at this point in obtaining the request token. Otherwise, the user interface will be thoroughly confusing.

This call returns an OAuth token:

```
oauth_token={token}&oauth_token_secret={secret}
```

The PHA is expected to store the Request Token and its correspondence to this specific user, likely in the web session.

**Authorize the Request Token**

Once it has obtained a request token, with the user’s browser still waiting for a response, the PHA responds by redirecting the user’s browser to the User Authorization URL on an Indivo UI app, indicated in the request token response above, or by default:

```
https://UI_SERVER/oauth/authorize?oauth_token=<REQUEST_TOKEN>
```

with the `request_token` as a URL query parameter named `oauth_token`. Note how this URL is not a signed OAuth request. This step is simply a redirection of the user’s browser to her Indivo account in order to prompt for and obtain authorization.

Indivo prompts the user to authenticate if she isn’t already logged in. Indivo then associates the request token with this user, and only this user can proceed with this specific request token. It is an error for a PHA to attempt to reuse request tokens, and Indivo will prevent this from happening.

Indivo then presents the user with the details of the PHA’s requested permissions.

The user can choose to cancel the process, in which case no further requests are issued, the PHA is not notified, and the request token is discarded.

**Obtain an Access Token**

If the user agrees to connect with the PHA, Indivo redirects the user browser to the PHA’s `callback_url`, as specified by the PHA at registration time. Appended to this `callback_url` are the `oauth_token`, the request token that identifies this authorization dance, and the `oauth_verifier`. The PHA is encouraged to check that the `oauth_token` matches the token stored in its web-session.

The PHA must now exchange the Request Token for an Access Token. This is accomplished using a 3-legged OAuth POST request, with the request token and secret, to:
In response to this request, the PHA obtains an Access Token, including one of two optional parameters:

```
oauth_token=<TOKEN>&oauth_token_secret=<SECRET>&xoauth_indivo_record_id=<RECORD_ID>
```

or

```
oauth_token=<TOKEN>&oauth_token_secret=<SECRET>&xoauth_indivo_carenet_id=<CARENET_ID>
```

This token can then be used by the PHA to make 3-legged OAuth calls to Indivo. The Indivo record ID parameter indicates which record this token is bound to, while the carenet indicates which portion of the system the PHA can access.

**Interact and Re-Auth** At this point, the PHA has an access token, an access secret, an Indivo record ID, and an Indivo privacy group. These credentials allow the PHA to make calls to the Indivo Server to obtain data from the given Indivo record. If the PHA provides a direct web interface to the user, this UI is delivered inside an IFRAME within the Indivo User Interface.

A few days later, when the user returns to his Indivo record, he can click on any of the PHAs he has already authorized. The PHA, however, does not know immediately who this user is. To communicate the user’s identity to the PHA, Indivo simply re-performs the OAuth dance, setting the IFRAME’s URL to the PHA’s starting point with the prescribed Indivo Record ID. When the PHA redirects the IFRAME to the authorization page, Indivo notices that this record has already authorized the app, and simply redirects the IFRAME immediately to the PHA’s callback_url. Thus, a complete OAuth process is re-performed, and the PHA re-obtains an access token, access secret, Indivo record ID and privacy group.

The PHA should never assume that the access token and secret stay the same. The long-term identifier that the PHA should key its data against is the Indivo Record ID.

**Admin Applications**

Admin Applications contact the Indivo X server using 2-legged OAuth only, with just a consumer key and consumer secret.

**Chrome Applications**

Most Indivo developers who only wish to write PHAs can safely ignore Chrome applications. Developers who wish to customize the entire Indivo experience need to understand Chrome apps.

The Indivo Chrome (User Interface) contacts the Indivo X server first using 2-legged OAuth to create a user-specific session using the user’s username and password. Indivo X responds with a fresh OAuth token and secret valid for the length of a typical web session. Then all Indivo Chrome calls to the Indivo X server on behalf of a given user are made as 3-legged calls, using the Indivo Chrome’s consumer key and secret, and the specific session token and secret.

**3.5.2 In-Browser Connect Authentication**

Connect-style authentication enables user applications running framed within the Indivo UI to make API calls solely using javascript, without having to navigate the OAuth dance. Connect-style authentication works as follows:

- When the Indivo UI app opens a user app within its iframe, it acquires a set of OAuth credentials that allows the UI app to make proxied API calls on behalf of the user app, using the API call POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials.
• The Indivo UI app additionally opens a channel to the iframe (using something like jschannel), so that the app can make client-side requests directly to the UI app.

• When the app wants to make an API call, it makes an unsigned request (in the client-side javascript) through its channel.

• The UI app receives the request, signs it with the Connect credentials, and passes the request through to the backend Indivo Server.

• Indivo Server processes the request, and sends the results back to the UI app, which in turn passes them through the channel to the user app.

Thus, the user app needs to perform no authentication when making API calls—it merely passes the calls through its channel to the UI app (a process which we’ve already implemented in our javascript client) and receives the results. The security of the call is enforced in the channel, and by the tokens used by the UI Server to authenticate the call.

### 3.5.3 Pre-generated REST Authentication

Pre-generated REST Authentication enables user applications running framed within the Indivo UI to acquire oAuth access tokens that can be used sign 3-legged API calls (as with standard Indivo oAuth) without having to navigate the oAuth dance. The authentication process is as follows:

• When the Indivo UI app opens a user app within its iframe, it acquires and preauthorizes an access token for the app, using the API call $\text{POST /accounts/\{ACCOUNT_EMAIL\}/apps/\{PHA_EMAIL\}/connect_credentials}$.

• The UI app then appends a well-formed oAuth header containing the access token to the user app’s start url. The format of the header (and required parameters) are described in the SMART documentation.

• The user app extracts the access token and indivo record ID from the oauth header, and uses it to sign subsequent API calls.

### 3.6 Indivo Basic Data Formats

#### 3.6.1 Timestamps

Using the standard XML Schema for timestamps in the UTC time zone, e.g. 2009-05-04T12:34:56Z.

#### 3.6.2 Null Values

Null elements are represented by removal of the element (deemed optional in the schema if nullable). Null attribute values are represented by the removal of the attribute (also deemed optional in the schema if nullable).

### 3.7 Indivo Client Libraries

An Indivo client library is any code (in any language) provided as a standard package to app developers which enables them to make API calls against Indivo without worrying about low-level implementation details such as OAuth signing. We currently support a few simple libraries: this document lists those libraries, and provides advice for generating new libraries.
3.7.1 Supported Client Libraries

Currently, we have released the following client libraries:

**Python Client**

Our python client is a simple wrapper around the commonly used and supported Python OAuth2 library for making OAuth-signed REST calls.

Our code is available on github, and documentation is available *in the Indivo docs.*

**iOS Framework**

The Indivo iOS Framework is an object-oriented wrapper that provides class-based access to core Indivo data-types using the Indivo API.

The code is available on github, and documentation is available *in the Indivo docs.*

**SMART Clients**

Since Indivo now supports the SMART API, apps written using SMART client libraries will also run on Indivo. See their [documentation](http://dev.smartplatforms.org) for more information.

**Note:** Make sure the the SMART clients you use support SMART v0.4

3.7.2 Building a Client Library

A client library’s responsibilities are simple: it must be able to sign HTTP requests using OAuth, send them to Indivo Server or an Indivo UI App (for OAuth authorization), and present the results of the requests back to the app developer. Most languages have libraries for doing these things already, so building a new Indivo client library is actually quite simple.

In order to facilitate auto-generation of clients, Indivo provides an *api.xml* file, which describes all of the calls a complete client should support. This file can be found in the indivo server source code, in `indivo_server/api.xml`.

The *api.xml* file should be updated whenever the supported API calls are modified. In order to insure that you have the most up-to-date version of the file, you can run (from a valid indivo_server checkout):

```
python manage.py generate_api_spec
```

Which will update the *api.xml* file to be consistent with the current codebase.

3.8 Coding Systems

For Indivo, and in general for a number of health applications, coding systems are used for interoperability. Examples include vaccine disease codes, allergy codes, procedure codes, etc. This page documents a web-based mechanism for documenting and publishing, in a machine-readable manner, these coding systems.
3.8.1 Abstract Model

Beyond the basic attributes (name, publisher, description), a coding system includes:

- a way to list all codes
- a way to look up a single code
- a way to search for codes matching a simple text query (e.g. “diab” should match “diabetes.”)

A single code entry will have, at least:

- a code
- an abbreviation
- a full title
- (optionally) a description
- (optionally) relationships to other codes.

3.8.2 Data Representation

JSON

3.8.3 RESTful Calls

The URL templates define RESTful calls to obtain a single code, and to search for a number of codes. Specifically, given the example above, the following URL returns a single code “123”:

http://codes.indivo.org/systems/allergies/123

And the following URL searches the list of allergy codes for “peanut”:

http://codes.indivo.org/systems/allergies/search?q=peanut

3.8.4 Sources

The coding systems used in Indivo X are as follows. Individual installations need to download the coding systems on their own, as the licenses for these do not permit redistribution, so we cannot package them with Indivo.

**Immunizations: HL7 v3**

The easiest way to get the HL7 V3 file in vertical-bar-separated format, as required by the coding system loader, is to use bioontology.org.

We specifically used the REST service at http://rest.bioontology.org/bioportal. The ontology code we used to download our version appears to no longer exist, so we’ll look into the latest codes soon. In the meantime, documentation for the REST service is at http://www.bioontology.org/wiki/index.php/NCBO_REST_services

**Labs: LOINC**

Available at http://loinc.org

There is an encoding issue which forced us to truncate the LOINC file for now at line 43504.
Problems: SNOMED CT

An encoding conversion is required to get to utf8, should be doable using the iconv program on most Linux installations.

Medications: RxTerms

Note that we may move to RxNorm instead of RxTerms.

3.9 API Query Interface

When a list of results are returned, the URL ends in a / and the HTTP method is a GET, as is typical of REST design. In that case, Indivo X supports a generic query string that determines paging and ordering of the results:

?offset={offset}&limit={limit}&order_by={order_by}&status={document_status}

• offset indicates which item number to start with, e.g. when getting a second batch of items.
• limit indicates the maximum number of items to return. This is used in combination with offset to accomplish paging.
• order_by is dependent on the fields returned in the list of items, and each call must thus define which fields are valid. Using an invalid field in order_by results in no effect on the output, as if order_by were absent.
• status can be used where applicable. It pertains to the status of documents and can currently be set to one of three options: ‘void’, ‘archived’ or ‘active’

For minimal (url ends in /reports/minimal/{report_type}/) and generic reports, we expose an expanded query interface, allowing for filtering, date constraints, and aggregation. Note that this interface only makes sense to implement for reports that function solely to retrieve and display fact objects. The interface will not be implemented for any more complex report types that deal with multiple fact objects (what would it mean for a CCR report to group by lab type?). Other reports will still have access to the Beta2-style paging operations. The interface will be available for other API calls to implement as well (i.e. the Audit interface)

3.9.1 Output

Minimal Reports

In the previous interface, reports were templated into schemas for output as specified by the Indivo Reporting Schema. This output method will be preserved for queries that return sets of fact objects, but for queries that return aggregates or groups, we will output data according to the Indivo Aggregate Report Schema.

Generic Reports

Non-Aggregate Generic Reports provide SDMJ or SDMX output based on the requested response format. Please see the documentation for response formats for more information.

Aggregate Also based on the requested response format

• XML
• JSON

Formatted as SDMJ with AggregateReport as the data model and without a __documentid__.
Below is an example of retrieving the max value using the date_group operator

```
[
    {
        "__modelname__": "AggregateReport",
        "group": "2009-07",
        "value": 1
    }, {
        "__modelname__": "AggregateReport",
        "group": "2009-08",
        "value": 4
    }
]
```

### 3.9.2 Data Fields

As in order_by in the Beta2 interface, each report must expose a set of data fields on which they may be filtered, grouped, or ordered. These fields are data-model dependent, and are explained below.

### 3.9.3 Query Operators

#### Filtering Operators

• offset, limit: Syntax is: `?offset={offset}&limit={limit}`. These operators will function as previously, taking integer indexes into the result set, and returning a sliced portion of the result set from indices offset to offset + limit.

• Custom Filters: Syntax is: `?{field}={value[|value]...}`. Limits result sets to items where the passed field is in the set of pipe delimited values. If no items have such a value in the passed field, the query will return an empty result set. Field names must be data fields exposed by the desired report type.

#### Ordering Operators

• order_by: Syntax is `?order_by={field}`. Functions as previously. Takes a data field exposed by the desired report type, and returns the result set sorted by that field. Fields are sorted in ascending order by default, and prefixing them with a '-' will reverse the order to descending.

  **Note:** If order_by is used with a grouping, {field} may only refer to the field used with group_by, date_group, or aggregate_by.

#### Grouping and Aggregating Operators

**Note:** Calls using grouping and aggregating operators will return data according to the aggregation schema, not the standard query schema.

• group_by: Syntax is `?group_by={field}`. Groups result sets by the passed field, which must be a data field exposed by the desired report type. **Must be used with an aggregation operator**, and will throw a 400 Bad Request error otherwise. **If used with order_by, the ordering field must be identical to the grouping field or the field passed in aggregate_by.**
• **aggregate_by**: Syntax is `?aggregate_by={operator}*{field}`. Combines multiple items in a result set (or a group within a result set) into a single item using the passed operator applied to the passed field (which must be a data field exposed by the desired report type). See below for examples. Available operators are:
  
  - **sum**: returns the sum of all values of `{field}`. Will throw a 400 Bad Request error if the passed field does not contain numerical data.
  
  - **avg**: returns the arithmetic mean of all values of `{field}`. Will throw a 400 Bad Request error if the passed field does not contain numerical data.
  
  - **max**: returns the maximum value of all values of `{field}`. Will throw a 400 Bad Request error if the passed field does not contain numerical data or date/time data.
  
  - **min**: returns the minimum value of all values of `{field}`. Will throw a 400 Bad Request error if the passed field does not contain numerical data or date/time data.
  
  - **count**: returns the total number of items passed. If `{field}` is specified, only counts rows where `<tt>{field}</tt>` is not empty.

  **Date-based Operators**

  • **date_range**: Syntax is `?date_range={field}*{start_date}*{end_date}`. A filtering operation that limits result sets to items with values of `{field}` between `{start_date}` and `{end_date}` (inclusive). If either `{start_date}` or `{end_date}` is not specified, the range will be open-ended. If both are unspecified, the filter will do nothing. `{field}` must be a data field exposed by the desired report type. If `{field}` is not a date/time field, a 400 Bad Request error will be raised. `{start_date}` and `{end_date}` must be entered as valid UTC timestamp strings, as described in **Basic Data Formats**. See below for examples.

  • **date_group**: Syntax is: `?date_group={field}*{time_increment}`. A grouping operator that, rather than grouping by a single field value, forms groups based on common increments of time. Has same restraints of use as **group_by** above, with the additional constraint that `{field}` must be a date/time data field.

    **If used with order_by, the ordering field must be identical to the grouping field or the field passed in aggregate_by.**

    **Note**: using this operator will result in the return of an aggregation schema.

    Valid increments are:

    - **hour**: items are placed in the same group if they occurred within the same hour.
    
    - **day**: items are placed in the same group if they occurred within the same day.
    
    - **week**: items are placed in the same group if they occurred within the same week.
    
    - **month**: items are placed in the same group if they occurred within the same month.
    
    - **year**: items are placed in the same group if they occurred within the same year.
    
    - **hourofday**: items are placed in the same group if they occurred during the same hour of day (even on separate days).
    
    - **dayofweek**: items are placed in the same group if they occurred on the same day of the week (even in separate weeks).
    
    - **weekofyear**: items are placed in the same group if they occurred during the same week of the year (indexed from 1 - 52), even in separate years.
    
    - **monthofyear**: items are placed in the same group if they occurred during the same month of the year (indexed from 1-12), even in separate years.
3.9.4 Query Operator Evaluation

Query operators are evaluated as follows:

1. filter operators, including `date_range` but excluding `limit` and `offset`, are applied first.
2. If `group_by` or `date_group` is passed, it is evaluated next.
3. `aggregate_by` is evaluated next.
4. `order_by`, `limit` and `offset` are applied.
5. The result set is templated into the standard schema or the aggregated schema as appropriate and returned.

3.9.5 Notes on Aggregation

Aggregation over Indivo medical data types could be very useful in certain cases where the data is known (by an app-developer, who generated the data, say) to be highly structured. For example, consider a ‘Pedometer-Visualizer’ app, which reads in data from an electric pedometer worn by a patient, stores that data as Indivo Measurements, and displays to the patient aggregate views of their steps taken (weekly/daily averages, total miles walked, etc.). This app could take full advantage of aggregation functions such as ‘sum’, ‘avg’, etc. However, there are many cases in Indivo where the data, in spite of conforming to Indivo schemas, is not necessarily clean enough to run these aggregations. Consider the case of lab test results: the schema field is by necessity a string, as not all lab results have numerical values. Thus, an incoming query might assume that it could ask for an ‘average lab result value’, when in fact the data wouldn’t support it. We therefore cannot allow numerical aggregations over fields not explicitly labeled as ‘Number’ types. If such a case is necessary for the app, the appropriate design is for the app to make a non-aggregate query, and then process the results itself (i.e., get all lab result values, and then do some data cleaning to insure that only relevant data points are counted in the averaging).

3.9.6 Default Operator Values

If omitted, the following query operators are assigned default values:

- `limit`: 100
- `offset`: 0
- `order_by`: ‘-created_at’ (the date when the fact object was added to indivo). **Only Applied to Non-aggregate Queries**: no default ordering for aggregate queries
- `status`: active

3.9.7 Valid Query Fields

With the new pluggable data models, valid query fields are defined by the data models themselves. See the Data Models documentation for a more complete explanation.

3.9.8 Example Queries

Below are a number of sample queries that demonstrate the power of the new interface.
Get all labs of type ‘Hematology’ within a date range

GET /records/{record_id}/reports/minimal/labs/?lab_type=Hematology&date_range=date_measured*2009-05-04T00:00:00Z*2011-03-09T00:00:00Z

Get all labs of type ‘Hematology’ or ‘Chemistry’

GET /records/{record_id}/reports/minimal/labs/?lab_type=Hematology|Chemistry

Get the number of lab results per type over the last year

GET /records/{record_id}/reports/minimal/labs/?group_by=lab_type&aggregate_by=count*lab_test_name&date_range=date_measured*2010-03-10T00:00:00Z*

Get the number of Hematology labs per month over the last year, ordered by date

GET /records/{record_id}/reports/minimal/labs/?lab_type=Hematology&date_group=date_measured*month&aggregate_by=count*lab_type&order_by=-date_measured&date_range=date_measured*2010-03-10T00:00:00Z*

3.10 Indivo Data Models

3.10.1 Introduction

Data Models in Indivo describe the format in which Indivo represents medical information. They are NOT the same as Schemas, which describe formats that Indivo recognizes as valid input data. Rather, data models describe the final processed state of medical data in Indivo: how data are stored, how they are queryable via the Query API, and how they are returned via the Reporting API.

We also introduce one additional term: Medical Facts. A Fact is one datapoint corresponding to a data model: for example, a latex allergy is a Fact that is an instance of the Allergy data model. Internally, Indivo represents facts as Python objects, so you’ll see us referencing medical facts as fact objects as well.

3.10.2 Defining a Data Model

At its most basic level, a data model definition is just a list of fields and their types. For example, our Problem data model is defined as (some fields omitted):

- date_onset: Date
- date_resolution: Date
- name: String
- comments: String
- diagnosed_by: String

This is pretty simple, and we’d like to enable others add new data models to Indivo just as easily. So we currently allow two formats for defining data models:
Django Model Classes

Since our data models are directly mapped to database tables using Django’s ORM, they are most effectively represented as Django Models. Django has a flexible, powerful method for expressing fields as python class attributes, so data models defined in this way can harness the full capabilities of the Django ORM. Of course, representing data models in this way requires some knowledge of python. For a full reference of Django models and Django model fields.

One important Indivo-specific note: when defining Django Model Classes, make sure to subclass `indivo.models.Fact`, which will ensure that your class can be treated as a data model. For example, your class definition might look like:

```python
from indivo.models import Fact
from django.db import models

class YourModel(Fact):
    your_field1 = models.CharField(max_length=200, null=True)
    ...

    # Additional fields here
```

Custom Django Model Fields

For modeling medical data, Indivo provides some custom Field Subclasses. These fields represent their data as multiple separate database fields, with names formed from the original field’s name and some appended suffixes (see the classes below for some examples). You should use these fields as if they were any other Django Model Field:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import YourFavoriteFieldSubclass

class YourModel(Fact):
    normal_field = models.CharField(max_length=200, null=True)
    special_field = YourFavoriteFieldSubclass()
```

Now YourModel has both a standard CharField, and also other fields defined by the Field Subclass. We define the following Field Subclasses:

```python
class indivo.fields.CodedValueField(Type)
A field for representing coded data elements.

    Creating a CodedValueField named ‘value’, for example, will (under the hood) create thee fields:
    •`value_identifier`, the system-specific identifier that represents the element (i.e. an RXNorm CUI)
    •`value_title`, the human-readable title of the element
    •`value_system`, the coding system used to represent the element

    When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `value` field name.

class indivo.fields.ValueAndUnitField(Type)
A field for representing data elements with both a value and a unit.

    Creating a ValueAndUnitField named ‘frequency’, for example, will (under the hood) create the fields:
    •`frequency_value`, the value of the element
```
• frequency_unit, the units in which the value is measured

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original frequency field name.

class indivo.fields.AddressField (Type)
A field for representing a physical address.

Creating an AddressField named ‘address’, for example, will (under the hood) create the fields:

• address_country, the country in which the address is located
• address_city, the city in which the address is located
• address_postalcode, the postalcode of the address
• address_region, the region (state, in the US) in which the address is located
• address_street, the street address (including street number, apartment number, etc.) at which the address is located

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original address field name.

class indivo.fields.NameField (Type)
A field for representing a person’s name.

Creating a NameField named ‘name’, for example, will (under the hood) create the fields:

• name_family, the family (last) name of the person
• name_given, the given (first) name of the person
• name_middle, the middle name of the person
• name_prefix, the prefix (i.e. ‘Mr.’, ‘Sir’, etc.) for the person’s name
• name_suffix, the suffix (i.e. ‘Jr.’, ‘Ph.D.’, etc.) for the person’s name

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original name field name.

class indivo.fields.TelephoneField (Type)
A field for representing a telephone number.

Creating a TelephoneField named ‘phone’, for example, will (under the hood) create the fields:

• phone_type, The type of the phone number, limited to h (home), w (work), or c (cell)
• phone_number, The actual phone number
• phone_preferred_p, Whether or not this number is a preferred method of contact (True or False)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original phone field name.

class indivo.fields.PharmacyField (Type)
A field for representing a pharmacy.

Creating a PharmacyField named ‘pharmacy’, for example, will (under the hood) create three fields:

• pharmacy_ncpdpid, the pharmacy’s National Council for Prescription Drug Programs (NCPDP) ID number
• pharmacy_adr, the address at which the pharmacy is located (an AddressField)
• pharmacy_org, the name of the organization that owns the pharmacy
When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original pharmacy field name.

```python
class Indivo.X.Documentation, Release 2.0.0

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original pharmacy field name.

```
Indivo X Documentation, Release 2.0.0

class indivo.fields.ValueRangeField (Type)
A field for representing a range of values.

Creating a ValueRangeField named ‘normal_range’, for example, will (under the hood) create the fields:

• normal_range_max, the maximum value of the range (a ValueAndUnitField)
• normal_range_min, the minimum value of the range (a ValueAndUnitField)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original normal_range field name.

class indivo.fields.QuantitativeResultField (Type)
A field for representing a quantitative result, and expected ranges for that result.

Creating a QuantitativeResultField named ‘lab_result’, for example, will (under the hood) create the fields:

• lab_result_non_critical_range, the range outside of which results are ‘critical’ (a ValueRangeField)
• lab_result_normal_range, the range outside of which results are ‘abnormal’ (a ValueRangeField)
• lab_result_value, the actual result (a ValueAndUnitField)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original lab_result field name.

Simple Data Modeling Language (SDML)

For those less python-savvy who are still capable of thinking in terms of ‘fields’ and ‘types’ (which should be most people), we’ve defined a JSON-based modeling language for defining the very simple data models easily. SDML is less flexible than Django’s modeling language, but is much quicker to get started with and is less verbose for describing simple models. See our documentation of the language here.

Feeling Lost?

For help getting started, see our core data models, below, each of which provide definitions both in SDML and Django Model classes.

3.10.3 Data Models and the Query API

Since the Query API allows app developers to directly apply filters and ranges to the datamodels they are selecting, they need to know what fields they are allowed to query against. The answer is simple:

ANY FIELD ON A DATA MODEL THAT IS NOT A RELATION TO ANOTHER MODEL MAY BE USED IN THE QUERY API!

For example, we introduced the ‘Problem’ model above, which has the fields:

• date_onset: Date
• date_resolution: Date
• name: String
• comments: String
• diagnosed_by: String

3.10.  Indivo Data Models
If you were making an API call such as `GET /records/(RECORD_ID)/reports/minimal/problems/`, you could filter by any of:

- `date_onset`
- `date_resolution`
- `name`
- `comments`
- `diagnosed_by`

If the problems model were a bit more complicated, and had another field:

- `prescribed_med`: Medication

You wouldn’t be able to filter by `prescribed_med`, since that field is a relation to another model.

The only exceptions to this rule are custom Django Model Fields. Such fields are translated into fields with other names, as described above. Any of these fields may be used in the query API, but (for example), when looking at a model with a CodedValue element such as:

- `problem_type`: CodedValue

You would be able to filter by `problem_type_identifier`, `problem_type_title`, or `problem_type_system`, but not by `problem_type` itself.

### 3.10.4 Core Data Models

Here is a listing of the data models currently supported by Indivo. Each instance might define other, contributed models: see below for information on how to add data models to Indivo.

**Indivo Data Model: Allergy**

**Model Definition**

As SDML:

```json
[
  {
    "__modelname__": "Allergy",
    "allergic_reaction": "CodedValue",
    "category": "CodedValue",
    "drug_allergen": "CodedValue",
    "drug_class_allergen": "CodedValue",
    "food_allergen": "CodedValue",
    "severity": "CodedValue"
  },
  {
    "__modelname__": "AllergyExclusion",
    "name": "CodedValue"
  }
]
```

As a Django Model Class:

```python
from indivo.models import Fact
from indivo.fields import CodedValueField

class Allergy(Fact):
    allergic_reaction = CodedValueField()
```
category = CodedValueField()
drug_allergen = CodedValueField()
drug_class_allergen = CodedValueField()
food_allergen = CodedValueField()
severity = CodedValueField()

class AllergyExclusion(Fact):
    name = CodedValueField()

Examples

As SDMJ:

```
[{
    "__modelname__": "Allergy",
    "allergic_reaction_title": "Anaphylaxis",
    "allergic_reaction_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
    "allergic_reaction_identifier": "39579001",
    "category_title": "Drug allergy",
    "category_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
    "category_identifier": "416098002",
    "drug_class_allergen_title": "Sulfonamide Antibacterial",
    "drug_class_allergen_system": "http://purl.bioontology.org/ontology/NDFRT/",
    "drug_class_allergen_identifier": "N0000175503",
    "severity_title": "Severe",
    "severity_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
    "severity_identifier": "24484000"
},
{
    "__modelname__": "AllergyExclusion",
    "name_title": "No known allergies",
    "name_identifier": "160244002",
    "name_system": "http://purl.bioontology.org/ontology/SNOMEDCT"
}]
```

As SDMX:

```
<Models>
  <Model name="Allergy">
    <Field name="allergic_reaction_title">Anaphylaxis</Field>
    <Field name="allergic_reaction_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="allergic_reaction_identifier">39579001</Field>
    <Field name="category_title">Drug allergy</Field>
    <Field name="category_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="category_identifier">416098002</Field>
    <Field name="drug_class_allergen_title">Sulfonamide Antibacterial</Field>
    <Field name="drug_class_allergen_system">http://purl.bioontology.org/ontology/NDFRT/</Field>
    <Field name="drug_class_allergen_identifier">N0000175503</Field>
    <Field name="severity_title">Severe</Field>
    <Field name="severity_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="severity_identifier">24484000</Field>
  </Model>
  <Model name="AllergyExclusion">
    <Field name="name_title">No known allergies</Field>
    <Field name="name_identifier">160244002</Field>
    <Field name="name_system">http://purl.bioontology.org/ontology/SNOMEDCT</Field>
  </Model>
</Models>
```
As a Fact object:

```python
from indivo.models import Allergy

allergy_fact = Allergy(
    allergic_reaction_title="Anaphylaxis",
    allergic_reaction_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    allergic_reaction_identifier="39579001",
    category_title="Drug allergy",
    category_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    category_identifier="416098002",
    drug_class_allergen_title="Sulfonamide Antibacterial",
    drug_class_allergen_system="http://purl.bioontology.org/ontology/NDFRT/",
    drug_class_allergen_identifier="N0000175503",
    severity_title="Severe",
    severity_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    severity_identifier="24484000",
)

allergy_exclusion = AllergyExclusion(
    name_title="No known allergies",
    name_identifier="160244002",
    name_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
)
```

**Indivo Data Model: Equipment**

**Model Definition**

As SDML:

```json
{
    "__modelname__": "Equipment",
    "date_started": "Date",
    "date_stopped": "Date",
    "name": "String",
    "vendor": "String",
    "description": "String"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class Equipment(Fact):
    date_started = models.DateField(null=True)
    date_stopped = models.DateField(null=True)
    name = models.CharField(max_length=40)
    vendor = models.CharField(max_length=40, null=True)
    description = models.TextField(null=True)
```
Examples

As SDMJ:

```json
{
    "__modelname__": "Equipment",
    "date_started": "2009-02-05",
    "date_stopped": "2009-06-12",
    "name": "Pacemaker",
    "vendor": "Acme Medical Devices",
    "description": "it works!"
}
```

As SDMX:

```xml
<Models>
    <Model name="Equipment">
        <Field name="date_started">2009-02-05</Field>
        <Field name="date_stopped">2009-06-12</Field>
        <Field name="name">Pacemaker</Field>
        <Field name="vendor">Acme Medical Devices</Field>
        <Field name="description">it works!</Field>
    </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import Equipment
from indivo.lib.iso8601 import parse_utc_date as date

equipment_fact = Equipment(
    date_started=date("2009-02-05"),
    date_stopped=date("2009-06-12"),
    name="Pacemaker",
    vendor="Acme Medical Devices",
    description="it works!"
)
```

Indivo Data Model: Immunization

Model Definition

As SDML:

```json
{
    "__modelname__": "Immunization",
    "date": "Date",
    "administration_status": "CodedValue",
    "product_class": "CodedValue",
    "product_class_2": "CodedValue",
    "product_name": "CodedValue",
    "refusal_reason": "CodedValue"
}
```

As a Django Model Class:
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField

class Immunization(Fact):
    date = models.DateTimeField(null=True)
    administration_status = CodedValueField()
    product_class = CodedValueField()
    product_class_2 = CodedValueField()
    product_name = CodedValueField()
    refusal_reason = CodedValueField()

Examples

As SDMJ:

{
    "__modelname__": "Immunization",
    "date": "2009-05-16T12:00:00Z",
    "administration_status_title": "Not Administered",
    "administration_status_system": "http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#",
    "administration_status_identifier": "notAdministered",
    "product_class_title": "TYPHOID",
    "product_class_system": "http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#",
    "product_class_identifier": "TYPHOID",
    "product_name_title": "typhoid, oral",
    "product_name_system": "http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx#",
    "product_name_identifier": "25",
    "refusal_reason_title": "Allergy to vaccine/vaccine components, or allergy to eggs",
    "refusal_reason_system": "http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#",
    "refusal_reason_identifier": "allergy"
}

As SDMX:

<Models>
    <Model name="Immunization">
        <Field name="date">2009-05-16T12:00:00Z</Field>
        <Field name="administration_status_title">Not Administered</Field>
        <Field name="administration_status_system">http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#</Field>
        <Field name="administration_status_identifier">notAdministered</Field>
        <Field name="product_class_system">http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#</Field>
        <Field name="product_class_identifier">TYPHOID</Field>
        <Field name="product_name_system">http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx#</Field>
        <Field name="product_name_identifier">25</Field>
        <Field name="refusal_reason_system">http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#</Field>
        <Field name="refusal_reason_identifier">allergy</Field>
    </Model>
</Models>

As a Fact object:

from indivo.models import Immunization
from indivo.lib.iso8601 import parse_utc_date as date
```python
immunization_fact = Immunization(
    date=date("2009-05-16T12:00:00Z"),
    administration_status_title="Not Administered",
    administration_status_system="http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#",
    administration_status_identifier="notAdministered",
    product_class_title="TYPHOID",
    product_class_system="http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vgf",
    product_class_identifier="TYPHOID",
    product_name_title="typhoid, oral",
    product_name_system="http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx",
    product_name_identifier="25",
    refusal_reason_title="Allergy to vaccine/vaccine components, or allergy to eggs",
    refusal_reason_system="http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#",
    refusal_reason_identifier="allergy",
)
```

**Indivo Data Model: LabResult**

**Model Definition**

As SDML:

```json
{
    "__modelname__": "LabResult",
    "abnormal_interpretation": "CodedValue",
    "accession_number": "String",
    "test_name": "CodedValue",
    "status": "CodedValue",
    "narrative_result": "String",
    "notes": "String",
    "quantitative_result": "QuantitativeResult",
    "collected_at": "Date",
    "collected_by_org": "Organization",
    "collected_by_name": "Name",
    "collected_by_role": "String"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField, QuantitativeResultField, OrganizationField, NameField

class LabResult(Fact):
    abnormal_interpretation = CodedValueField()
    accession_number = models.CharField(max_length=255, null=True)
    test_name = CodedValueField()
    status = CodedValueField()
    narrative_result = models.CharField(max_length=255, null=True)
    notes = models.CharField(max_length=600, null=True)
    quantitative_result = QuantitativeResultField()
    collected_at = models.DateTimeField(null=True)
    collected_by_org = OrganizationField()
    collected_by_name = NameField()
    collected_by_role = models.CharField(max_length=255, null=True)
```
Examples

As SDMJ:

```json
{
    "__modelname__": "LabResult",
    "abnormal_interpretation_title": "Normal",
    "abnormal_interpretation_system": "http://smartplatforms.org/terms/codes/LabResultInterpretation#",
    "abnormal_interpretation_identifier": "normal",

    "accession_number": "AC09205823577",

    "test_name_title": "Serum Sodium",
    "test_name_system": "http://purl.bioontology.org/ontology/LNC/",
    "test_name_identifier": "2951-2",

    "status_title": "Final results": complete and verified",
    "status_system": "http://smartplatforms.org/terms/codes/LabStatus#",
    "status_identifier": "final",

    "notes": "Blood sample appears to have hemolyzed",

    "quantitative_result_non_critical_range_max_value": "155",
    "quantitative_result_non_critical_range_max_unit": "mEq/L",
    "quantitative_result_non_critical_range_min_value": "120",
    "quantitative_result_non_critical_range_min_unit": "mEq/L",

    "quantitative_result_normal_range_max_value": "145",
    "quantitative_result_normal_range_max_unit": "mEq/L",
    "quantitative_result_normal_range_min_value": "135",
    "quantitative_result_normal_range_min_unit": "mEq/L",

    "quantitative_result_value_value": "140",
    "quantitative_result_value_unit": "mEq/L",

    "collected_at": "2010-12-27T17:00:00Z",

    "collected_by_org_name": "City Lab",
    "collected_by_org adr_country": "USA",
    "collected_by_org adr_city": "Springfield",
    "collected_by_org adr_postalcode": "11111",
    "collected_by_org adr_region": "MA",
    "collected_by_org adr street": "20 Elm St",

    "collected_by_name_family": "Finnialispi",
    "collected_by_name_given": "Tad",

    "collected_by_role": "Lab Specialist"
}
```

As SDMX:

```xml
<Models>
    <Model name="LabResult">
        <Field name="abnormal_interpretation_title">Normal</Field>
        <Field name="abnormal_interpretation_system">http://smartplatforms.org/terms/codes/LabResultInterpretation#</Field>
        <Field name="abnormal_interpretation_identifier">normal</Field>

        <Field name="accession_number">AC09205823577</Field>
    </Model>
</Models>
```
<Field name="test_name_title">Serum Sodium</Field>
<Field name="test_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="test_name_identifier">2951-2</Field>

<Field name="status_title">Final results: complete and verified</Field>
<Field name="status_system">http://smartplatforms.org/terms/codes/LabStatus#</Field>
<Field name="status_identifier">final</Field>

<Field name="notes">Blood sample appears to have hemolyzed</Field>
<Field name="quantitative_result_non_critical_range_max_value">155</Field>
<Field name="quantitative_result_non_critical_range_max_unit">mEq/L</Field>
<Field name="quantitative_result_non_critical_range_min_value">120</Field>
<Field name="quantitative_result_non_critical_range_min_unit">mEq/L</Field>

<Field name="quantitative_result_normal_range_max_value">145</Field>
<Field name="quantitative_result_normal_range_max_unit">mEq/L</Field>
<Field name="quantitative_result_normal_range_min_value">135</Field>
<Field name="quantitative_result_normal_range_min_unit">mEq/L</Field>

<Field name="quantitative_result_value_value">140</Field>
<Field name="quantitative_result_value_unit">mEq/L</Field>
<Field name="collected_at">2010-12-27T17:00:00Z</Field>
<Field name="collected_by_org_name">City Lab</Field>
<Field name="collected_by_org_adr_country">USA</Field>
<Field name="collected_by_org_adr_city">Springfield</Field>
<Field name="collected_by_org_adr_postalcode">11111</Field>
<Field name="collected_by_org_adr_region">MA</Field>
<Field name="collected_by_org_adr_street">20 Elm St</Field>
<Field name="collected_by_name_family">Finnialispi</Field>
<Field name="collected_by_name_given">Tad</Field>
<Field name="collected_by_role">Lab Specialist</Field>

As a Fact object:

```python
from indivo.models import LabResult
from indivo.lib.iso8601 import parse_utc_date as date

lab_fact = LabResult(
    abnormal_interpretation_title="Normal",
    abnormal_interpretation_system="http://smartplatforms.org/terms/codes/LabResultInterpretation#",
    abnormal_interpretation_identifier="normal",
    accession_number="AC09205823577",
    test_name_title="Serum Sodium",
    test_name_system="http://purl.bioontology.org/ontology/LNC/",
    test_name_identifier="2951-2",
    status_title="Final results: complete and verified",
    status_system="http://smartplatforms.org/terms/codes/LabStatus#",
    status_identifier="final",
)```

3.10. Indivo Data Models
notes="Blood sample appears to have hemolyzed",
quantitative_result_non_critical_range_max_value="155",
quantitative_result_non_critical_range_max_unit="mEq/L",
quantitative_result_non_critical_range_min_value="120",
quantitative_result_non_critical_range_min_unit="mEq/L",
quantitative_result_normal_range_max_value="145",
quantitative_result_normal_range_max_unit="mEq/L",
quantitative_result_normal_range_min_value="135",
quantitative_result_normal_range_min_unit="mEq/L",
quantitative_result_value_value="140",
quantitative_result_value_unit="mEq/L",
collected_at=date("2010-12-27T17:00:00Z"),
collected_by_org_name="City Lab",
collected_by_org adr_country="USA",
collected_by_org adr_city="Springfield",
collected_by_org adr_postalcode="11111",
collected_by_org adr_region="MA",
collected_by_org adr_street="20 Elm St",
collected_by_name_family="Finnialispi",
collected_by_name_given="Tad",
collected_by_role="Lab Specialist",
)

**Indivo Data Model: Medication**

**Model Definition**

As *SDML*:

```
{  
  "__modelname__": "Medication",
  "drugName": "CodedValue",
  "endDate": "Date",
  "frequency": "ValueAndUnit",
  "instructions": "String",
  "provenance": "CodedValue",
  "quantity": "ValueAndUnit",
  "startDate": "Date",
  "fulfillments": [{
    "__modelname__": "Fill",
    "date": "Date",
    "dispenseDaysSupply": "Number",
    "pbm": "String",
    "pharmacy": "Pharmacy",
    "provider": "Provider",
    "quantityDispensed": "ValueAndUnit"
  }]
}
```
Note: Since SDML doesn’t provide for Boolean Fields, we are unable to define the `dispense_as_written` field properly in SDML. Our actual implementation of the Medication data model uses a Django Model Class for this reason.

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField, ValueAndUnitField, PharmacyField, ProviderField

class Medication(Fact):
    drugName = CodedValueField()
    endDate = models.DateField(null=True)
    frequency = ValueAndUnitField()
    instructions = models.CharField(max_length=255, null=True)
    provenance = CodedValueField()
    quantity = ValueAndUnitField()
    startDate = models.DateField(null=True)

class Fill(Fact):
    date = models.DateTimeField(null=True)
    dispenseDaysSupply = models.FloatField(null=True)
    pbm = models.CharField(max_length=255, null=True)
    pharmacy = PharmacyField()
    provider = ProviderField()
    quantityDispensed = ValueAndUnitField()
    medication = models.ForeignKey(Medication, null=True, related_name='fulfillments')
```

Examples

As SDMJ:

```json
{
    "__modelname__": "Medication",
    "drugName_title": "AMITRIPTYLINE HCL 50 MG TAB",
    "drugName_system": "http://purl.bioontology.org/ontology/RXNORM/",
    "drugName_identifier": "856845",
    "endDate": "2007-08-14",
    "frequency_value": "2",
    "frequency_unit": "/d",
    "instructions": "Take two tablets twice daily as needed for pain",
    "provenance_title": "Derived by prescription",
    "provenance_system": "http://smartplatforms.org/terms/codes/MedicationProvenance#",
    "provenance_identifier": "prescription",
    "quantity_value": "2",
    "quantity_unit": "(tablet)",
    "startDate": "2007-03-14",
    "fulfillments": [
        {
            "__modelname__": "Fill",
            "date": "2007-03-14T04:00:00Z",
            "dispenseDaysSupply": "30",
            "pbm": "T000000000001011",
            "pharmacy_ncpdpid": "5235235",
            "pharmacy_org": "CVS #588",
            "pharmacy_adr_country": "Australia",
            "pharmacy_adr_city": "WonderCity",
            "pharmacy_adr_postalcode": "5555",
            "pharmacy_adr_street": "111 Lake Drive",
        }
    ]
}
```
As SDMX:

```
<Models>
  <Model name="Medication">
    <Field name="drugName_title">AMITRIPTYLINE HCL 50 MG TAB</Field>
    <Field name="drugName_system">http://purl.bioontology.org/ontology/RXNORM/</Field>
    <Field name="drugName_identifier">856845</Field>
    <Field name="endDate">2007-08-14</Field>
    <Field name="frequency_value">2</Field>
    <Field name="frequency_unit">/d</Field>
    <Field name="instructions">Take two tablets twice daily as needed for pain</Field>
    <Field name="provenance_title">Derived by prescription</Field>
    <Field name="provenance_system">http://smartplatforms.org/terms/codes/MedicationProvenance#</Field>
    <Field name="provenance_identifier">prescription</Field>
    <Field name="quantity_value">2</Field>
    <Field name="quantity_unit">{tablet}</Field>
    <Field name="startDate">2007-03-14</Field>
  </Model>
  <Models>
    <Model name="Fill">
      <Field name="date">2007-03-14T04:00:00Z</Field>
      <Field name="dispenseDaysSupply">30</Field>
    </Model>
  </Models>
```

Chapter 3. App Developers
As a Fact object:

```python
from indivo.models import Medication, Fill
from indivo.lib.iso8601 import parse_utc_date as date

med = Medication(
    drugName_title="AMITRIPTYLINE HCL 50 MG TAB",
    drugName_system="http://purl.bioontology.org/ontology/RXNORM/",
    drugName_identifier="856845",
    endDate=date("2007-08-14"),
    frequency_value="2",
    frequency_unit="/d",
    instructions="Take two tablets twice daily as needed for pain",
    provenance_title="Derived by prescription",
    provenance_system="http://smartplatforms.org/terms/codes/MedicationProvenance#",
)```
provenance_identifier="prescription",
quantity_value="2",
quantity_unit="{tablet}",
startDate=date("2007-03-14")
)

fill1 = Fill(
    date=date("2007-03-14T04:00:00Z"),
dispenseDaysSupply=30,
    pbm="T0000000001011",
    pharmacy_ncpdpid="5235235",
    pharmacy_org="CVS #588",
    pharmacy adr country="Australia",
    pharmacy adr city="WonderCity",
    pharmacy adr postalcode="5555",
    pharmacy adr street="111 Lake Drive",
    provider_dea_number="325555555",
    provider_npi_number="5235235",
    provider_email="joshua.mandel@fake.emailserver.com",
    provider name given="Josuha",
    provider name family="Mandel",
    provider tel 1 type="w",
    provider tel 1 number="1-235-947-3452",
    provider tel 1 preferred_p=True,
    quantityDispensed_value="60",
    quantityDispensed_unit="{tablet}"
)

fill2 = Fill(
    date=date("2007-04-14T04:00:00Z"),
dispenseDaysSupply=30,
    pbm="T0000000001011",
    pharmacy_ncpdpid="5235235",
    pharmacy_org="CVS #588",
    pharmacy adr country="Australia",
    pharmacy adr city="WonderCity",
    pharmacy adr postalcode="5555",
    pharmacy adr street="111 Lake Drive",
    provider_dea_number="325555555",
    provider_npi_number="5235235",
    provider_email="joshua.mandel@fake.emailserver.com",
    provider name given="Josuha",
    provider name family="Mandel",
    provider tel 1 type="w",
    provider tel 1 number="1-235-947-3452",
    provider tel 1 preferred_p=True,
    quantityDispensed_value="60",
    quantityDispensed_unit="{tablet}"
)

# save the medication so we can relate other objects to it
med.save()
med.fulfillments = [fill1, fill2]
med.save()
Indivo Data Model: Problem

Model Definition

As SDML:

```json
{
    "__modelname__": "Problem",
    "startDate": "Date",
    "endDate": "Date",
    "name": "CodedValue",
    "notes": "String"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from indivo.fields import CodedValueField
from django.db import models

class Problem(Fact):
    startDate = models.DateTimeField(null=True)
    endDate = models.DateTimeField(null=True)
    name = CodedValueField()
    notes = models.TextField(null=True)
```

Examples

As SDMJ:

```json
{
    "__modelname__": "Problem",
    "startDate": "2009-05-16T12:00:00Z",
    "endDate": "2009-05-16T16:00:00Z",
    "name_title": "Backache (finding)",
    "name_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
    "name_identifier": "161891005"
}
```

As SDMX:

```xml
<Models>
  <Model name="Problem">
    <Field name="startDate">2009-05-16T12:00:00Z</Field>
    <Field name="endDate">2009-05-16T16:00:00Z</Field>
    <Field name="name_title">Backache (finding)</Field>
    <Field name="name_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="name_identifier">161891005</Field>
  </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import Problem
from indivo.lib.iso8601 import parse_utc_date as date

problem_fact = Problem(
    startDate=date("2009-05-16T12:00:00Z"),
)```
endDate=date("2009-05-16T16:00:00Z"),
name_title="Backache (finding)",
name_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
name_identifier="161891005",
)

Indivo Data Model: Procedure

Model Definition

As SDML:

```json
{
    "__modelname__": "Procedure",
    "date_performed": "Date",
    "name": "String",
    "name_type": "String",
    "name_value": "String",
    "name_abbrev": "String",
    "provider_name": "String",
    "provider_institution": "String",
    "location": "String",
    "comments": "String"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
class Procedure(Fact):
    date_performed = models.DateTimeField(null=True)
    name = models.CharField(max_length=100)
    name_type = models.CharField(max_length=80, null=True)
    name_value = models.CharField(max_length=40, null=True)
    name_abbrev = models.CharField(max_length=20, null=True)
    provider_name = models.CharField(max_length=200, null=True)
    provider_institution = models.CharField(max_length=200, null=True)
    location = models.CharField(max_length=100, null=True)
    comments = models.TextField(null=True)
```

Examples

As SDMJ:

```json
{
    "__modelname__": "Procedure",
    "date_performed": "2009-05-16T12:00:00",
    "name": "Appendectomy",
    "name_type": "http://codes.indivo.org/procedures#",
    "name_value": "123",
    "name_abbrev": "append",
    "provider_name": "Kenneth Mandl",
    "provider_institution": "Children’s Hospital Boston",
    "location": "300 Longwood Ave, Boston MA 02115",
    "```
As SDMX:

```xml
<Models>
  <Model name="Procedure">
    <Field name="date_performed">2009-05-16T12:00:00</Field>
    <Field name="name">Appendectomy</Field>
    <Field name="name_type">http://codes.indivo.org/procedures#</Field>
    <Field name="name_value">123</Field>
    <Field name="name_abbrev">append</Field>
    <Field name="provider_name">Kenneth Mandl</Field>
    <Field name="provider_institution">Children’s Hospital Boston</Field>
    <Field name="location">300 Longwood Ave, Boston MA 02115</Field>
    <Field name="comments">Went great!</Field>
  </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import Procedure
from indivo.lib.iso8601 import parse_utc_date as date

procedure_fact = Procedure(
    date_performed=date("2009-05-16T12:00:00"),
    name="Appendectomy",
    name_type="http://codes.indivo.org/procedures#",
    name_value="123",
    name_abbrev="append",
    provider_name="Kenneth Mandl",
    provider_institution="Children’s Hospital Boston",
    location="300 Longwood Ave, Boston MA 02115",
    comments="Went great!"
)
```

**Indivo Data Model: VitalSigns**

**Model Definition**

As SDML:

```json
{
  "__modelname__": "VitalSigns",
  "date": "Date",
  "encounter": {
    "__modelname__": "Encounter",
    "startDate": "Date",
    "endDate": "Date",
    "facility": "Organization",
    "provider": "Provider",
    "encounterType": "CodedValue"
  },
  "bp": "BloodPressure",
  "bmi": "VitalSign",
  "heart_rate": "VitalSign",
  "height": "VitalSign",
}
```
As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import BloodPressureField, VitalSignField, CodedValueField, OrganizationField, ProviderField

class VitalSigns(Fact):
    date = models.DateTimeField(null=True)
    encounter = models.ForeignKey('Encounter', null=True)
    bp = BloodPressureField()
    bmi = VitalSignField()
    heart_rate = VitalSignField()
    height = VitalSignField()
    oxygen_saturation = VitalSignField()
    respiratory_rate = VitalSignField()
    temperature = VitalSignField()
    weight = VitalSignField()

class Encounter(Fact):
    startDate = models.DateTimeField(null=True)
    endDate = models.DateTimeField(null=True)
    facility = OrganizationField()
    provider = ProviderField()
    encounterType = CodedValueField()
```

Examples

As SDMJ:

```json
{
    "__modelname__": "VitalSigns",
    "date": "2009-05-16T12:00:00Z",
    "encounter": {
        "__modelname__": "Encounter",
        "startDate": "2009-05-16T12:00:00Z",
        "endDate": "2009-05-16T16:00:00Z",
        "facility_name": "Wonder Hospital",
        "facility_adr_country": "Australia",
        "facility_adr_city": "WonderCity",
        "facility_adr_postalcode": "5555",
        "facility_adr_street": "111 Lake Drive",
        "provider_dea_number": "325555555",
        "provider_npi_number": "5235235",
        "provider_email": "joshua.mandel@fake.emailserver.com",
        "provider_name_given": "Josuha",
        "provider_name_family": "Mandel",
        "provider_tel_1_type": "w",
        "provider_tel_1_number": "1-235-947-3452",
        "provider_tel_1_preferred_p": true,
        "encounterType_title": "Ambulatory encounter",
        "encounterType_system": "http://smartplatforms.org/terms/codes/EncounterType#",
    }
```
"encounterType_identifier": "ambulatory"
],
"bp_position_title": "Sitting",
"bp_position_identifier": "33586001",
"bp_position_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
"bp_site_title": "Right arm",
"bp_site_identifier": "368209003",
"bp_site_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
"bp_method_title": "Auscultation",
"bp_method_identifier": "auscultation",
"bp_method_system": "http://smartplatforms.org/terms/codes/BloodPressureMethod#",
"bp_diastolic_unit": "mm[Hg]",
"bp_diastolic_value": 82,
"bp_diastolic_name_title": "Intravascular diastolic",
"bp_diastolic_name_identifier": "8462-4",
"bp_diastolic_name_system": "http://purl.bioontology.org/ontology/LNC/",
"bp_systolic_unit": "mm[Hg]",
"bp_systolic_value": 132,
"bp_systolic_name_title": "Intravascular systolic",
"bp_systolic_name_identifier": "8480-6",
"bp_systolic_name_system": "http://purl.bioontology.org/ontology/LNC/",
"bmi_unit": "kg/m2",
"bmi_value": 21.8,
"bmi_name_title": "Body mass index",
"bmi_name_system": "http://purl.bioontology.org/ontology/LNC/",
"bmi_name_identifier": "39156-5",
"heart_rate_unit": "(beats)/min",
"heart_rate_value": 70,
"heart_rate_name_title": "Heart rate",
"heart_rate_name_system": "http://purl.bioontology.org/ontology/LNC/",
"heart_rate_name_identifier": "8867-4",
"height_unit": "m",
"height_value": 1.8,
"height_name_title": "Body height",
"height_name_system": "http://purl.bioontology.org/ontology/LNC/",
"height_name_identifier": "8302-2",
"oxygen_saturation_unit": "%{HemoglobinSaturation}"
,"oxygen_saturation_value": 99,
"oxygen_saturation_name_title": "Oxygen saturation",
"oxygen_saturation_name_system": "http://purl.bioontology.org/ontology/LNC/",
"oxygen_saturation_name_identifier": "2710-2",
"respiratory_rate_unit": "(breaths)/min",
"respiratory_rate_value": 16,
"respiratory_rate_name_title": "Respiration rate",
"respiratory_rate_name_system": "http://purl.bioontology.org/ontology/LNC/",
"respiratory_rate_name_identifier": "9279-1",
"temperature_unit": "Cel",
"temperature_value": 37,
"temperature_name_title": "Body temperature",
"temperature_name_system": "http://purl.bioontology.org/ontology/LNC/",
"temperature_name_identifier": "8310-5"
"weight_unit": "kg",
"weight_value": 70.8,
"weight_name_title": "Body weight",
"weight_name_system": "http://purl.bioontology.org/ontology/LNC/",
"weight_name_identifier": "3141-9"
}

As SDMX:

<Models>
    <Model name="VitalSigns">
        <Field name="date">2009-05-16T12:00:00Z</Field>
        <Field name="encounter">
            <Model name="Encounter">
                <Field name="startDate">2009-05-16T12:00:00Z</Field>
                <Field name="endDate">2009-05-16T16:00:00Z</Field>
                <Field name="facility_name">Wonder Hospital</Field>
                <Field name="facility_adr_country">Australia</Field>
                <Field name="facility_adr_city">WonderCity</Field>
                <Field name="facility_adr_postalcode">5555</Field>
                <Field name="facility_adr_street">111 Lake Drive</Field>
                <Field name="provider_dea_number">325555555</Field>
                <Field name="provider_npi_number">5235235</Field>
                <Field name="provider_email">joshua.mandel@fake.emailserver.com</Field>
                <Field name="provider_name_given">Josuha</Field>
                <Field name="provider_name_family">Mandel</Field>
                <Field name="provider_tel_1_type">w</Field>
                <Field name="provider_tel_1_number">1-235-947-3452</Field>
                <Field name="provider_tel_1_preferred_p">true</Field>
                <Field name="encounterType_title">Ambulatory encounter</Field>
                <Field name="encounterType_system">http://smartplatforms.org/terms/codes/EncounterType#</Field>
                <Field name="encounterType_identifier">ambulatory</Field>
            </Model>
        </Field>
        <Field name="bp_position_title">Sitting</Field>
        <Field name="bp_position_identifier">33586001</Field>
        <Field name="bp_site_title">Right arm</Field>
        <Field name="bp_site_identifier">368209003</Field>
        <Field name="bp_method_title">Auscultation</Field>
        <Field name="bp_method_identifier">auscultation</Field>
        <Field name="bp_diastolic_unit">mm[Hg]</Field>
        <Field name="bp_diastolic_value">82</Field>
        <Field name="bp_diastolic_name_title">Intravascular diastolic</Field>
        <Field name="bp_diastolic_name_identifier">8462-4</Field>
        <Field name="bp_systolic_unit">mm[Hg]</Field>
        <Field name="bp_systolic_value">132</Field>
        <Field name="bp_systolic_name_title">Intravascular systolic</Field>
        <Field name="bp_systolic_name_identifier">8480-6</Field>
        <Field name="bmi_unit">kg/m2</Field>
        <Field name="bmi_value">21.8</Field>
        <Field name="bmi_name_title">Body mass index</Field>
        <Field name="bmi_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
        <Field name="bmi_name_identifier">39156-5</Field>
    </Model>
</Models>
<Field name="heart_rate_unit">{beats}/min</Field>
<Field name="heart_rate_value">70</Field>
<Field name="heart_rate_name_title">Heart rate</Field>
<Field name="heart_rate_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="heart_rate_name_identifier">8867-4</Field>
<Field name="height_unit">m</Field>
<Field name="height_value">1.8</Field>
<Field name="height_name_title">Body height</Field>
<Field name="height_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="height_name_identifier">8302-2</Field>
<Field name="oxygen_saturation_unit">%{HemoglobinSaturation}</Field>
<Field name="oxygen_saturation_value">99</Field>
<Field name="oxygen_saturation_name_title">Oxygen saturation</Field>
<Field name="oxygen_saturation_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="oxygen_saturation_name_identifier">2710-2</Field>
<Field name="respiratory_rate_unit">{breaths}/min</Field>
<Field name="respiratory_rate_value">16</Field>
<Field name="respiratory_rate_name_title">Respiration rate</Field>
<Field name="respiratory_rate_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="respiratory_rate_name_identifier">9279-1</Field>
<Field name="temperature_unit">Cel</Field>
<Field name="temperature_value">37</Field>
<Field name="temperature_name_title">Body temperature</Field>
<Field name="temperature_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="temperature_name_identifier">8310-5</Field>
<Field name="weight_unit">kg</Field>
<Field name="weight_value">70.8</Field>
<Field name="weight_name_title">Body weight</Field>
<Field name="weight_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="weight_name_identifier">3141-9</Field>

As a Fact object:

```python
from indivo.models import Encounter, VitalSigns
from indivo.lib.iso8601 import parse_utc_date as date

encounter_fact = Encounter(
    startDate=date("2009-05-16T12:00:00Z"),
    endDate=date("2009-05-16T16:00:00Z"),
    facility_name="Wonder Hospital",
    facility_adr_country="Australia",
    facility_adr_city="WonderCity",
    facility_adr_postalcode="5555",
    facility_adr_street="111 Lake Drive",
    provider_dea_number="325555555",
    provider_npi_number="5235235",
    provider_email="joshua.mandel@fake.emailserver.com",
    provider_name_given="Josuha",
    provider_name_family="Mandel",
    provider_tel_1_type="w",
    provider_tel_1_number="1-235-947-3452",
    provider_tel_1_preferred_p=True,
    encounterType_title="Ambulatory encounter",
    encounterType_system="http://smartplatforms.org/terms/codes/EncounterType#",
    encounterType_identifier="ambulatory",
)
encounter_fact.save()
```

3.10. Indivo Data Models 239
# NOTE: all vitals readings are OPTIONAL. You don’t need
# to add all 56 fields here to create a VitalSigns object.
vitals_fact = VitalSigns(
    date=date("2009-05-16T12:00:00Z"),
    encounter=encounter_fact,

    # Blood Pressure
    bp_position_title="Sitting",
    bp_position_identifier="33586001",
    bp_position_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    bp_site_title="Right arm",
    bp_site_identifier="368209003",
    bp_site_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    bp_method_title="Auscultation",
    bp_method_identifier="auscultation",
    bp_method_system="http://smartplatforms.org/terms/codes/BloodPressureMethod#",
    bp_diastolic_unit="mm[Hg]",
    bp_diastolic_value=82,
    bp_diastolic_name_title="Intravascular diastolic",
    bp_diastolic_name_identifier="8462-4",
    bp_diastolic_name_system="http://purl.bioontology.org/ontology/LNC/",
    bp_systolic_unit="mm[Hg]",
    bp_systolic_value=132,
    bp_systolic_name_title="Intravascular systolic",
    bp_systolic_name_identifier="8480-6",
    bp_systolic_name_system="http://purl.bioontology.org/ontology/LNC/",

    # Body Mass Index
    bmi_unit="kg/m2",
    bmi_value=21.8,
    bmi_name_title="Body mass index",
    bmi_name_system="http://purl.bioontology.org/ontology/LNC/",
    bmi_name_identifier="39156-5",

    # Heart Rate
    heart_rate_unit="{beats}/min",
    heart_rate_value=70,
    heart_rate_name_title="Heart rate",
    heart_rate_name_system="http://purl.bioontology.org/ontology/LNC/",
    heart_rate_name_identifier="8867-4",

    # Height
    height_unit="m",
    height_value=1.8,
    height_name_title="Body height",
    height_name_system="http://purl.bioontology.org/ontology/LNC/",
    height_name_identifier="8302-2",

    # Oxygen Saturation
    oxygen_saturation_unit="\%{HemoglobinSaturation}"
    oxygen_saturation_value=99,
    oxygen_saturation_name_title="Oxygen saturation",
    oxygen_saturation_name_system="http://purl.bioontology.org/ontology/LNC/",
    oxygen_saturation_name_identifier="2710-2",

    # Respiratory Rate
    respiratory_rate_unit="{breaths}/min",
    respiratory_rate_value=16,
Indivo X Documentation, Release 2.0.0

Indivo Data Model: Simple_Clinical_Note

Model Definition

As SDML:

```json
{  
  "__modelname__": "SimpleClinicalNote",
  "date_of_visit": "Date",
  "finalized_at": "Date",
  "visit_type": "String",
  "visit_type_type": "String",
  "visit_type_value": "String",
  "visit_type_abbrev": "String",
  "visit_location": "String",
  "specialty": "String",
  "specialty_type": "String",
  "specialty_value": "String",
  "specialty_abbrev": "String",
  "signed_at": "Date",
  "provider_name": "String",
  "provider_institution": "String",
  "chief_complaint": "String",
  "content": "String"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class SimpleClinicalNote(Fact):
    date_of_visit = models.DateTimeField()
    finalized_at = models.DateTimeField(null=True)
    visit_type = models.CharField(null=True, max_length=100)
    visit_type_type = models.CharField(max_length=80, null=True)
    visit_type_value = models.CharField(max_length=40, null=True)
    visit_type_abbrev = models.CharField(max_length=20, null=True)
```
visit_location = models.CharField(max_length=200, null=True)
specialty = models.CharField(null=True, max_length=100)
specialty_type = models.CharField(max_length=80, null=True)
specialty_value = models.CharField(max_length=40, null=True)
specialty_abbrev = models.CharField(max_length=20, null=True)
signed_at = models.DateTimeField(null=True)
provider_name = models.CharField(null=True, max_length=200)
provider_institution = models.CharField(max_length=200, null=True)
chief_complaint = models.CharField(null=True, max_length=255)
content = models.TextField(null=True)

Examples

As SDMJ:

```
{
    "__modelname__": "SimpleClinicalNote",
    "date_of_visit": "2010-02-02T12:00:00Z",
    "finalized_at": "2010-02-03T13:12:00Z",
    "visit_type": "Acute Care",
    "visit_type_type": "http://codes.indivo.org/visit-types#",
    "visit_type_value": "123",
    "visit_type_abbrev": "acute",
    "visit_location": "Longfellow Medical",
    "specialty": "Hematology/Oncology",
    "specialty_type": "http://codes.indivo.org/specialties#",
    "specialty_value": "234",
    "specialty_abbrev": "hem-onc",
    "signed_at": "2010-02-03T13:12:00Z",
    "provider_name": "Kenneth Mandl",
    "provider_institution": "Children's Hospital Boston",
    "chief_complaint": "stomach ache",
    "content": "Patient presents with ..."
}
```

As SDMX:

```
<Models>
    <Model name="SimpleClinicalNote">
        <Field name="date_of_visit">2010-02-02T12:00:00Z</Field>
        <Field name="finalized_at">2010-02-03T13:12:00Z</Field>
        <Field name="visit_type">Acute Care</Field>
        <Field name="visit_type_type">http://codes.indivo.org/visit-types#</Field>
        <Field name="visit_type_value">123</Field>
        <Field name="visit_type_abbrev">acute</Field>
        <Field name="visit_location">Longfellow Medical</Field>
        <Field name="specialty">Hematology/Oncology</Field>
        <Field name="specialty_type">http://codes.indivo.org/specialties#</Field>
        <Field name="specialty_value">234</Field>
        <Field name="specialty_abbrev">hem-onc</Field>
        <Field name="signed_at">2010-02-03T13:12:00Z</Field>
        <Field name="provider_name">Kenneth Mandl</Field>
        <Field name="provider_institution">Children's Hospital Boston</Field>
        <Field name="chief_complaint">stomach ache</Field>
        <Field name="content">Patient presents with ...</Field>
    </Model>
</Models>
```
As a Fact object:

```python
from indivo.models import SimpleClinicalNote
from indivo.lib.iso8601 import parse_utc_date as date

simple_clinical_note_fact = SimpleClinicalNote(
    date_of_visit=date("2010-02-02T12:00:00Z"),
    finalized_at=date("2010-02-03T13:12:00Z"),
    visit_type="Acute Care",
    visit_type_type="http://codes.indivo.org/visit-types#",
    visit_type_value="123",
    visit_type_abbrev="acute",
    visit_location="Longfellow Medical",
    specialty="Hematology/Oncology",
    specialty_type="http://codes.indivo.org/specialties#",
    specialty_value="234",
    specialty_abbrev="hem-onc",
    signed_at=date("2010-02-03T13:12:00Z"),
    provider_name="Kenneth Mandl",
    provider_institution="Children’s Hospital Boston",
    chief_complaint="stomach ache",
    content="Patient presents with ..."
)
```

### 3.10.5 Advanced Data-Model Tasks

#### Adding Advanced Features to a Data-Model

For complicated data models, a simple SDML definition just won’t suffice. For a few specific features, such as custom object serialization or creation-time field validation, you can define (in python) an extra options file for a data model.

This file should be named `extra.py`, and can be dropped into the filesystem next to any data model, as described below. The file should contain subclasses of `indivo.data_models.options.DataModelOptions`, each of which describes the options for one data model defined in the `model.py` file in the same directory. Options are:

```python
class indivo.data_models.options.DataModelOptions (Type)
    Defines optional extra functionality for Indivo datamodels.

    To add options to a datamodel, subclass this class and override its attributes.
```

Currently available options are:

- **model_class_name**: **Required**. The name of the datamodel class to attach to.

- **serializers**: Custom serializers for the data model. Should be set to a subclass of `indivo.serializers.DataModelSerializers`.

- **field_validators**: Custom validators for fields on the data model. A dictionary, where keys are field names on the model, and values are lists of Django Validators to be run against the field.

For example, here’s our options file for the Problem data model:

```python
from indivo.serializers import DataModelSerializers
from indivo.data_models.options import DataModelOptions
from indivo.validators import ExactValueValidator

SNOMED_URI = ‘http://purl.bioontology.org/ontology/SNOMEDCT/’

class ProblemSerializers(DataModelSerializers):
```
def to_rdf(queryset, result_count, record=None, carenet=None):
    # ... our SMART RDF serializer implementation here ...#
    return 'some RDF'

class ProblemOptions(DataModelOptions):
    model_class_name = 'Problem'
    serializers = ProblemSerializers
    field_validators = {
        'name_system': [ExactValueValidator(SNOMED_URI)],
    }

Make sure to restart Indivo for your changes to take effect after you add your `extra.py` file—but there’s no need to reset Indivo.

Adding Custom Serializers to a Data-Model

By default, when returning data via the `generic reporting API`, Indivo will attempt to serialize data as SDMJ or SDMX, depending on the requested response format. If you need your data to come back in other formats, or if the default serializers aren’t smart enough to represent your data model correctly, you can implement custom serializers for the data model.

Defining the Serializers

Serializers for a data model are implemented as simple methods that take a Django queryset object, and return a serialized string. For a given data-model, you should define a subclass of `indivo.serializers.DataModelSerializers`, and add your desired serializers as methods on the class. Currently, available serializers are:

- `to_xml(queryset, result_count, record=None, carenet=None)`
  returns an XML string representing the model objects in `queryset`.
  
  Parameters
  - `queryset (QuerySet)` – the objects to serialize
  - `result_count` (`integer`) – the total number of items in `queryset`
  - `record` (`Record`) – the patient record that the objects belong to, if available.
  - `carenet` (`Carenet`) – the Carenet via which the objects have been retrieved, if available.
  
  Return type string

- `to_json(queryset, result_count, record=None, carenet=None)`
  returns a JSON string representing the model objects in `queryset`.

  Parameters
  - `queryset (QuerySet)` – the objects to serialize
  - `result_count` (`integer`) – the total number of items in `queryset`
  - `record` (`Record`) – the patient record that the objects belong to, if available.
  - `carenet` (`Carenet`) – the Carenet via which the objects have been retrieved, if available.

  Return type string

- `to_rdf(queryset, result_count, record=None, carenet=None)`
  returns an RDF/XML string representing the model objects in `queryset`. 
Indivo X Documentation, Release 2.0.0

Parameters
• queryset (QuerySet) – the objects to serialize
• result_count (integer) – the total number of items in queryset
• record (Record) – the patient record that the objects belong to, if available.
• carenet (Carenet) – the Carenet via which the objects have been retrieved, if available.
Return type string
For example, here’s a (non-functional) implementation of the serializers for the Problems data-model:
from indivo.serializers import DataModelSerializers
class ProblemSerializers(DataModelSerializers):
def to_xml(queryset, result_count, record=None, carenet=None):
return ’’’<Problems>...bunch of problems here...</Problems>’’’
def to_json(queryset, result_count, record=None, carenet=None):
return ’’’[{"Problem": "data here"}, {"Problem": "More data here..."}]’’’

def to_rdf(queryset, result_count, record=None, carenet=None):
return ’’’<rdf:RDF><rdf:Description rdf:type=’indivo:Problem’>...RDF data here...</rdf:Descri

A couple things to note:
• The to_*() methods DO NOT take self as their first argument. Under the hood, we actually rip the methods
out of the serializers class and attach them directly to the data-model class.
• The model_class_name attribute is required, and indicates which data-model the serializers should be attached to.
Libraries for Serialization

When serializing models, the following libraries can come in handy:
• lxml.etree: Our favorite XML manipulation library. See http://lxml.de/tutorial.html for the details. Lxml
is required for a running Indivo instance, so it will always be available for import (from lxml import
etree).
Django bundles a version of simplejson, which can be imported with from django.utils import
simplejson.
may not be installed on all systems, so if you use it, make sure to install it first.
Attaching the Serializers to a Data Model

Adding custom serializers to a data-model is simple: simply set your DataModelSerializers subclass to the
serializers attribute of a DataModelOptions subclass in an extra.py file (see above for info on adding
advanced data-model options.
Adding Field Validation to a Data-Model
By default, data models defined in SDML are very permissive: all fields are nullable, and there are no constraints on
valid data points other than their type (string, date, etc.). In some cases, a data element could satisfy these constraints,
3.10. Indivo Data Models

245


but still be invalid. For example, an Indivo Problem must have its name coded using SNOMED, so a problem without a snomed code is invalid.

**Defining the Validators**

In such cases, you can attach validators to the data model. Django Validators are essentially just python callables that raise a `django.core.exceptions.ValidationError` if they are called on an invalid data point. We’ve defined a couple of useful validators, though you could use any function you’d like.

For example, here’s a validator that will accept only the value 2:

```python
from django.core.exceptions import ValidationError

def validate_2(value):
    if value != 2:
        raise ValidationError("Invalid value: \$s. Expected 2"%str(value))
```

**Built in Validators**

Django provides a number of built-in validators, for which a full reference exists here: https://docs.djangoproject.com/en/1.2/ref/validators/#built-in-validators.

In addition, Indivo defines a few useful validators in `indivo.validators`:

```python
class indivo.validators.ValueInSetValidator(valid_values, nullable=False)
    Validates that a value is within a set of possible values.
    
    The optional ‘nullable’ flag determines whether or not the value may also be empty.

class indivo.validators.ExactValueValidator(valid_value, nullable=False)
    Validates that a value is exactly equal to a certain value.
    
    The optional ‘nullable’ flag determines whether or not the value may also be empty.
```

**Attaching Validators to a Data Model**

Adding custom validators to a data-model is simple: simply add the validator to the field_validators attribute of a `DataModelOptions` subclass in an `extra.py` file (see above for info on adding advanced data-model options).

For example, let’s add the requirement that Problem names must be coded as snomed. We can write the validator using the built-in `ExactValueValidator`:

```python
from indivo.validators import ExactValueValidator
SNOMED_URI = 'http://purl.bioontology.org/ontology/SNOMEDCT/
snomed_validator = ExactValueValidator(SNOMED_URI)
```

We can then attach it to the `name_system` field of a Problem, which will guarantee that we only accept problems which identify themselves as having a snomed code for their names:

```python
class ProblemOptions(DataModelOptions):
    model_class_name = 'Problem'
    field_validators = {
        'name_system': [snomed_validator]
    }
```

Note that we put `snomed_validator` in a list, since we might theoretically add additional validators to the `name_system` field.
Adding Custom Data-Models to Indivo

As of version 1.1 of Indivo X, we’ve added a feature that makes it much easier to add (in a drag-and-drop fashion) new supported data models to an instance of Indivo. Adding a new data model to Indivo involves:

- Creating the data model definition
- Dropping the data model into the filesystem
- Migrating the database tables to support the new model

Defining the Data Model

As you saw above, data models can be defined in two formats: SDML or Django model classes. Simply produce a definition in one of the two forms, and save it to a file named `model.sdml` or `model.py`.

Dropping the Definition into the Filesystem

Indivo data models currently have the following layout on the filesystem:

```
indivo_server/
    indivo/
        ...
        data_models/
            core/
                allergy/
                    model.[sdml | py]
                    example.[sdmj | sdmx | py]
                    extra.py
            ...
        contrib/
```

The `indivo/data_models/core/` directory contains all of our built-in data models, and you shouldn’t modify it. Since you are ‘contributing’ a data model to Indivo, add your data model to the `indivo/data_models/contrib/` directory. Simply:

- Create a new subdirectory under `indivo/data_models/contrib/`.
- Drop your model definition into that directory. This file MUST BE NAMED `MODEL.PY` OR `MODEL.SDML` to be identified as a data model.
- Add (optional) example files into that directory. Files should be named `example.sdmj`, `example.sdmx`, or `example.py`, and should be example instances of the data model as SDMJ, SDMX, or Fact objects respectively. If present, they will help others use and document your data model.
- Add an (optional) extras file to the directory. The file must be named `extra.py`, and may contain extra options for your data-model, such as custom serializers.
- Your final directory structure should now look something like:

```
indivo_server/
    indivo/
        ...
        data_models/
            core/
                allergy/
                    model.[sdml | py]
                    example.[sdmj | sdmx | py]
```

3.10. Indivo Data Models
Migrating the Database

Indivo relies on the South migration tool to get the database synced with the latest data models. Once you’ve dropped your data model into the filesystem, South should be able to detect the necessary changes.

To detect the new model and generate migrations for it, run (from the indivo_server directory):

```bash
python manage.py schemamigration indivo --auto
```

You should see output like:

```
+ Added model indivo.YOURMODELNAME
Created 0018_auto__add_model_YOURMODELNAME.py. You can now apply this migration with: ./manage.py migrate indivo
```

To do a quick sanity check that you aren’t about to blow away your database, run:

```bash
python manage.py migrate indivo --db-dry-run -v2
```

This should output the SQL that will be run. Make sure this looks reasonable, ESPECIALLY if you are running Indivo on Oracle, where the South tool is still in alpha. If the SQL looks reasonable, go ahead and run the migration, with:

```bash
python manage.py migrate indivo
```

And you’re all set!

Next Steps

Make sure to restart Indivo for your changes to take effect.

See also:

Now you’ve added a new data model to Indivo: Congratulations! It can be stored in the database and queried via the API.

But until you map a Schema to it, you won’t be able to actually add data to your new model. To learn more, see:

- Data Pipeline
- Adding Custom Schemas to Indivo
- Adding Custom Transforms to Indivo
3.11 Data Pipeline

3.11.1 Introduction

The Indivo Data Pipeline is the set of processes that take input data as it enters Indivo, extract and store clinical datapoints (Facts), and make those datapoints available as output via the API. As of version 1.1 of Indivo X, every component of the pipeline is substitutable: you can add new formats for input data, new ways to store the data, and new methods to output them.

Let’s start with some vocabulary:

**Schema** A description of a format for data, which can also be used to validate that data. For example, our SDMX Schema defines the format of incoming data in our SDMX specification language, and can be used to determine whether an XML document is valid SDMX. In Indivo, we use schemas to define the formats in which we accept data via the API, and validate that data as it comes in. Right now, schemas can only take the form of XSDs, since we accept input data only in XML form, but in the future this might change. More on Indivo schemas here.

**Document** A collection of data, formatted according to a Schema. When you use our API to add data to Indivo, the data you send to Indivo constitutes one document.

**Transform** A tool that takes as input some data which validates against a schema, and outputs it in a format consistent with Indivo’s Data Models. We currently accept two types of transforms: XSLTs and Python code. We also understand several output formats for the data after it has been transformed. More on transforms in Indivo here.

**Data Model** A definition of the format in which processed data is stored in Indivo’s database. Each ‘data model’ corresponds to a type of clinical information. For example, our Medication Data Model describes how we represent a processed medication in the database. We currently accept two types of data model definitions. More on Indivo Data Models here.

**Fact** A single datapoint: an instance of a data model. For example, one Medication is a Fact, as is one Problem, etc. The medication fact is an object whose format is defined by the Medication Data Model. When you take input data in the form of an XML Document and run it through a Transform, you arrive at a Fact object that can be stored in the database.

**Serialization** The process of taking a Fact object and converting it to an output format, suitable for returning in response to an API call. For example, our current Reporting API outputs Fact objects serialized to XML or JSON.

3.11.2 The Pipeline

As you may have surmised from the above definitions, the data pipeline in Indivo is actually pretty simple, and consists of five steps:

1. **Identification.** An incoming document is examined, and its type determined (right now, since data comes only in XML documents, a document’s type can be uniquely constructed from it’s root nodename and namespace, i.e. http://indivo.org/vocab/xml/documents#Allergy).
2. **Validation.** The identified document is matched against its schema. If it is invalid, the pipeline terminates, and an error is thrown.
3. **Transformation.** The validated document is then processed using its transform. If the output matches one of the valid transform output formats, it is converted into one or more Fact objects, ready for storage.
4. **Storage.** The processed Facts are written to the database.
5. **Retrieval.** When an app makes an API call using the Reporting API, the database is queried for matching Facts. Those facts are then serialized into the required output format (i.e., XML or JSON) and returned to the app.
Figure 3.1: The Indivo Data Pipeline, Visualized
With the new data pipeline, *Schemas*, *Transforms*, and *Data Models* are all substitutable: you can add and replace them at will.

Notice that there is not a one-to-one relationship between incoming documents and processed fact objects. This allows for Indivo to accept schemas like a CCR, which contains many facts. A parsed CCR document might end up outputting many Problem, Medication, Allergy, Lab, or other facts, even though there was only one input document.

Conversely, there is not one document type associated with producing one type of fact object. This allows Indivo to accept the same type of data in many formats. For example, you could get a medication fact from our standard medication document, but you could also get a medication fact from a CCR.

The data pipeline is activated whenever new data is added to Indivo using document creation API calls, and whenever data is retrieved from Indivo using the reporting API calls. The following calls add new documents to Indivo, and therefore feed data into the pipeline:

- POST /records/{RECORD_ID}/documents/
- PUT /records/{RECORD_ID}/documents/external/{APP_ID}/{EXTERNAL_ID}
- POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace
- PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace/external/{APP_ID}/{EXTERNAL_ID}
- POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/
- PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/external/{APP_ID}/{EXTERNAL_ID}
- POST /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}/accept

All of the *reporting* calls retrieve collections of processed Fact objects from Indivo, and thus rely on the tail end of the processing pipeline.

### 3.11.3 Learning More

See also:

More information on the components of the data pipeline can be found here:

- *Indivo Schemas*
- *Indivo Transforms*
- *Indivo Data Models*

More information on customizing the data pipeline can be found here:

- *Adding Custom Schemas to Indivo*
- *Adding Custom Transforms to Indivo*
- *Adding Custom Data-Models to Indivo*

#### 3.12 Indivo Experimental APIs

The Indivo experimental APIs are used to evaluate new potential features. They are not guaranteed to be supported, consistent, or used in production. When an API is experimental, it is under the `/experimental/` URL base. An API graduates to full support by being moved into the normal tree of API URLs, which requires a change in all clients.
that implement it. This is done on purpose: to ensure that experimental APIs cannot be depended upon, ever, until they graduate to full support.

### 3.12.1 PubSub

*Status: first design, no implementation*

Applications may want to be notified of changes that occur in the record, so they can take action without having to constantly poll the Indivo server. We call these subscriptions.

#### Add a subscription

```
POST /experimental/records/{record_id}/apps/{pha_email}/subscriptions/
url={url}&
callback={callback_url}

<Subscription id="..." />
```

A subscription is created by an app on a specific URL, within a record. The URL is relative to the record, i.e. /reports/minimal/medications/. For now only report URLs are supported.

The callback URL is the URL that Indivo should connect to when an update pertaining to that subscription occurs.

#### View subscriptions

```
GET /experimental/records/{record_id}/apps/{pha_email}/subscriptions/

<Subscriptions>
  <Subscription id="..">
    <expires_at>...</expires_at>
    <url>...</url>
  </Subscription>
</Subscriptions>
```

A subscription lasts 3 months by default, and then automatically expires and must be renewed.

#### Remove subscription

```
DELETE /experimental/records/{record_id}/apps/{pha_email}/subscriptions/{subscription_id}
```

#### Behavior of a subscription ping

When an app is to be notified of a change, if, for example, that app subscribes to medications, then a new medication would trigger:

```
POST <callback_url>
record_id={record_id} &
url={url}
```

Pings based on a subscription are expect a 200 successful HTTP code. If they receive a 500 or other error code, the ping will be tried again with exponential backoff, using 1 minute as initial retry interval, and a factor of 2 between retries.
3.13 Messaging and Notifications

In Indivo X, apps and users can message or notify each other. This page describes the various means of communication.

3.13.1 Messaging

Messaging in Indivo is very much like email: every message has a subject and a body.

Records vs. Accounts and Routing

Are messages sent to a record, or to an account? In Indivo X, messages can be sent to either one, but they always end up in an account’s inbox (records don’t have inboxes). In other words, if an app sends a message to a record, it is routed to a set of accounts with a note indicating which record it pertains to. The rules for routing record messages are, for now simple: the owner of the record and any account that has full access to that record receive a copy of a message sent to that record.

Message Content

Messages have a subject, a body, and a body type, which indicates the formatting of the body. (The subject can only be plain text.) The two body types supported at this time are plaintext and markdown. Plaintext is what it sounds like: text that is shown as is, without any HTML interpretation. Markdown is a format that allows for simple markup features like bold, italics, and links, without all the complexity (and security complications) of HTML.

Specifics of Markdown

Markdown support in Indivo is in so-called “safe mode,” where any HTML markup is silently stripped from the markdown content, and only markdown syntax is transformed into HTML bold, italics, and simple links. All links open up in new browser windows, except for links that are meant to send the user to the app that sent the message. Specifically, with the following markdown syntax:

Go [back to the FDA app]({APP_BASE}/message?id={MESSAGE_ID}) for more information.

Notice the URL contains standard URL template variable notation, using curly brackets. The variables {APP_BASE} and {MESSAGE_ID} are substituted appropriately, and because this is a link to an app, clicking it will not open a new window, but rather send the user to that app, with the given URL.

Messaging a Record

Messaging an Account

3.13.2 Notifications

Notifications only pertain to records.
3.14 Python Client Library Reference

class indivo_python_client.IndivoClient(server_params, consumer_params, re-
source_token=None, **state_vars)

The Indivo python client. The client should be initialized with the following arguments:

- **server_params**: A dictionary containing information about the API server. The dictionary should have two keys: `api_base`, the server location from which the API is served, and `authorization_base`, the server location to which the user should be redirected to perform OAuth authorization.

- **consumer_params**: A dictionary containing information about your app. The dictionary should have two keys: `consumer_key`, the OAuth consumer key for your app, and `consumer_secret`, the OAuth consumer secret for your app.

- **resource_token**: Optional. A token (request, access, or session) with which to sign requests. The token should be a dictionary with two keys: `oauth_token` and `oauth_token_secret`.

- **state_vars**: Optional. Additional state to track with the API. This will be used to fill in url parameters when available. For example, if you create an IndivoClient with `pha_email='myapp@apps.indivo.org'`, then making an API call like GET /apps/{PHA_EMAIL} will automatically fill in the url, without you needing to pass the `pha_email` argument into the call. This is useful with variables that are unlikely to change within the use of a single client object, such as `pha_email` or `record_id`. You can override the variables in individual API calls if need be, however.

IndivoClient.get(uri, body={}, headers={}, **uri_params)

Make a signed HTTP GET request against Indivo. Arguments are:

- **uri**: The URI against which to make the request. Optionally, the URI may have templatatable parameters, which should take the form of `{VAR_NAME}`. Such variables must be passed in via the `uri_params` argument or be present as a state variable on the `IndivoClient` instance itself, or a `KeyError` will be raised.

- **body**: Optional. A dictionary containing querystring parameters to add to the request, for example: `{ 'record_id': 'abcde' }`.

- **headers**: Optional. A dictionary containing additional HTTP headers to add to the request.

- **uri_params**: Optional. Additional parameters to be templated into the uri. Parameter names should be the lower-cased equivalent of uri parameters. For example, if the URI contains `{PHA_EMAIL}`, `uri_params` should contain `pha_email='someapp@apps.indivo.org'`.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

IndivoClient.put(uri, body='', headers={}, content_type=None, **uri_params)

Make a signed HTTP PUT request against Indivo. Arguments are:

- **uri**: The URI against which to make the request. Optionally, the URI may have templatatable parameters, which should take the form of `{VAR_NAME}`. Such variables must be passed in via the `uri_params` argument or be present as a state variable on the `IndivoClient` instance itself, or a `KeyError` will be raised.

- **body**: Optional. The body of the request. It should be either a raw data string, or a dictionary containing form-data parameters.

- **headers**: Optional. A dictionary containing additional HTTP headers to add to the request.
**content_type**: Optional. The MIME type of the data submitted in the PUT request. defaults to application/x-www-form-urlencoded.

**uri_params**: Optional. Additional parameters to be templated into the uri. parameter names should be the lower-cased equivalent of uri parameters. For example, if the URI contains `{PHA_EMAIL}`, `uri_params` should contain pha_email='someapp@apps.indivo.org'.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

```python
IndivoClient.post(\uri, body='', headers={}, content_type=None, **uri_params)
```
Make a signed HTTP POST request against Indivo. Arguments are:

- **uri**: The URI against which to make the request. Optionally, the URI may have templatable parameters, which should take the form of `{VAR_NAME}`. Such variables must be passed in via the `uri_params` argument or be present as a state variable on the IndivoClient instance itself, or a KeyError will be raised.
- **body**: Optional. The body of the request. It should be either a raw data string, or a dictionary containing form-data parameters.
- **headers**: Optional. A dictionary containing additional HTTP headers to add to the request.
- **content_type**: Optional. The MIME type of the data submitted in the POST request. defaults to application/x-www-form-urlencoded.
- **uri_params**: Optional. Additional parameters to be templated into the uri. parameter names should be the lower-cased equivalent of uri parameters. For example, if the URI contains `{PHA_EMAIL}`, `uri_params` should contain pha_email='someapp@apps.indivo.org'.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

```python
IndivoClient.delete(uri, headers=\{}, **uri_params)
```
Make a signed HTTP DELETE request against Indivo. Arguments are:

- **uri**: The URI against which to make the request. Optionally, the URI may have templatable parameters, which should take the form of `{VAR_NAME}`. Such variables must be passed in via the `uri_params` argument or be present as a state variable on the IndivoClient instance itself, or a KeyError will be raised.
- **headers**: Optional. A dictionary containing additional HTTP headers to add to the request.
- **uri_params**: Optional. Additional parameters to be templated into the uri. parameter names should be the lower-cased equivalent of uri parameters. For example, if the URI contains `{PHA_EMAIL}`, `uri_params` should contain pha_email='someapp@apps.indivo.org'.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

```python
IndivoClient.update_token(resource_token)
```
Update the token used by the client to sign requests. `resource_token` should be a dictionary with two keys: `oauth_token` and `oauth_token_secret`.

Returns None.
IndivoClient.fetch_request_token(params={})
Get a new request token from the server. params should include parameters for generating the token, such as indivo_record_id.

Returns the request token in the form of a dictionary with two keys: oauth_token and oauth_token_secret.

IndivoClient.exchange_token(verifier)
Exchange the client’s current token (a request token) for an access token. verifier must be the verifier string returned after the user has successfully authenticated.

Returns the newly acquired access token in the form of a dictionary with two keys: oauth_token and oauth_token_secret.

IndivoClient.get_surl_credentials()
Generate a token and secret for signing URLs. This token/secret are based on the client’s current resource token (which should be an access token). SURL credentials are required in order to use a UI Server widget: they delegate access to the UI Server to make API calls on behalf of a user app.

Returns a dictionary with two keys: token and secret.

IndivoClient.account_create([body={}, headers={}, content_type=None])
Create a new account, and send out initialization emails.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_search([body={}, headers={}])
Search for accounts by name or email.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/search for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_info(account_email=None, [body={}, headers={}])
Display information about an account.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.get_connect_credentials(account_email=None, pha_email=None[, body={}, headers={}, content_type=None])

Get OAuth credentials for an app to run in Connect or SMART REST mode.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.delete_user_preferences(account_email=None, pha_email=None[, headers={}])

Delete all app-specific User Preferences for an account.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.get_user_preferences(account_email=None, pha_email=None[, body={}, headers={}])

Get app-specific User Preferences for an account.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.set_user_preferences(account_email=None, pha_email=None[, body={}, headers={}, content_type=None])

Set app-specific User Preferences for an account.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_authsystem_add(account_email=None[, body={}, headers={}, content_type=None])

Add a new method of authentication to an account.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/authsystems/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.account_password_change(account_email=None, body={}, headers={}, content_type=None)

Change a account’s password.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/authsystems/password/change for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_password_set(account_email=None, body={}, headers={}, content_type=None)

Force the password of an account to a given value.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/authsystems/password/set for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_username_set(account_email=None, body={}, headers={}, content_type=None)

Force the username of an account to a given value.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/authsystems/password/set-username for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_check_secrets(account_email=None, primary_secret=None, body={}, headers={})

Validate an account’s primary and secondary secrets.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/check-secrets/{PRIMARY_SECRET} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_forgot_password(account_email=None, body={}, headers={}, content_type=None)

Resets an account if the user has forgotten its password.
**body** should contain the data to send with the call. **headers** should contain any additional HTTP headers to pass in with the call. **content_type** should contain the MIME type of the data sent in the **body** argument. See POST /accounts/{ACCOUNT_EMAIL}/forgot-password for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

---

**IndivoClient.account_inbox(account_email=None[, body={}, headers={}])**

List messages in an account’s inbox.

**body** should contain query parameters for the call. **headers** should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/inbox/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

---

**IndivoClient.account_send_message(account_email=None[, body={}, headers={}, content_type=None])**

Send a message to an account.

**body** should contain the data to send with the call. **headers** should contain any additional HTTP headers to pass in with the call. **content_type** should contain the MIME type of the data sent in the **body** argument. See POST /accounts/{ACCOUNT_EMAIL}/inbox/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

---

**IndivoClient.account_inbox_message(account_email=None, message_id=None[, body={}, headers={}])**

Retrieve an individual message from an account’s inbox.

**body** should contain query parameters for the call. **headers** should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/inbox/{MESSAGE_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

---

**IndivoClient.account_message_archive(account_email=None, message_id=None[, body={}, headers={}, content_type=None])**

Archive a message.

**body** should contain the data to send with the call. **headers** should contain any additional HTTP headers to pass in with the call. **content_type** should contain the MIME type of the data sent in the **body** argument. See POST /accounts/{ACCOUNT_EMAIL}/inbox/{MESSAGE_ID}/archive for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.account_inbox_message_attachment_accept(attachment_num=None, account_email=None, message_id=None, body={}, headers={}, content_type=None)

Accept a message attachment into the record it corresponds to.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}/accept for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_info_set(account_email=None, body={}, headers={}, content_type=None)

Set basic information about an account.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/info-set for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_initialize(account_email=None, primary_secret=None, body={}, headers={}, content_type=None)

Initialize an account, activating it.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/initialize/{PRIMARY_SECRET} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_notifications(account_email=None, body={}, headers={})

List an account’s notifications.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/notifications/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_permissions(account_email=None, body={}, headers={})

List the carenets that an account has access to.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/permissions/ for valid parameters and the returned content.
This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_primary_secret(account_email=None[, body={}, headers={}])
Display an account’s primary secret.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/primary-secret for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_list(account_email=None[, body={}, headers={}])
List all available records for an account.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/records/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_reset(account_email=None[, body={}, headers={}, content_type=None])
Reset an account to an uninitialized state.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/{ACCOUNT_EMAIL}/reset for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_secret(account_email=None[, body={}, headers={}])
Return the secondary secret of an account.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/secret for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_resend_secret(account_email=None[, body={}, headers={}, content_type=None])
Sends an account user their primary secret in case they lost it.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/{ACCOUNT_EMAIL}/secret-resend for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
**IndivoClient.account_set_state**

```python
account_set_state(account_email=None, body={}, headers={}, content_type=None)
```

Set the state of an account.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/set-state for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

**IndivoClient.all_phas**

```python
all_phas(body={}, headers={})
```

List all available userapps.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

**IndivoClient.all_manifests**

```python
all_manifests(body={}, headers={})
```

List SMART manifests for all available userapps.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/manifests/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

**IndivoClient.pha_delete**

```python
pha_delete(pha_email=None, headers={})
```

Delete a userapp from Indivo.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

**IndivoClient.pha**

```python
pha(pha_email=None, body={}, headers={})
```

Return a description of a single userapp.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.
IndivoClient.app_document_list(pha_email=None, body={}, headers={})

List app-specific documents.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/{PHA_EMAIL}/documents/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.app_document_create(pha_email=None, body={}, headers={}, content_type=None)

Create an app-specific Indivo document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /apps/{PHA_EMAIL}/documents/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.app_document_create_or_update_ext(external_id=None,pha_email=None, body={}, headers={}, content_type=None)

Create an app-specific Indivo document with an associated external id.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.app_document_meta_ext(external_id=None,pha_email=None, body={}, headers={})

Fetch the metadata of an app-specific document identified by external id.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.app_document_delete(pha_email=None, document_id=None, headers={})

Delete an app-specific document.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.app_specific_document (pha_email=None, document_id=None[, body={}, headers={}])

Retrive an app-specific document.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.app_document_create_or_update (pha_email=None, document_id=None[, body={}, headers={}, content_type=None])

Create or Overwrite an app-specific Indivo document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.app_document_label (pha_email=None, document_id=None[, body={}, headers={}, content_type=None])

Set the label of an app-specific document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/label for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.app_document_meta (pha_email=None, document_id=None[, body={}, headers={}])

Fetch the metadata of an app-specific document.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.app_manifest (pha_email=None[, body={}, headers={}])

Return a SMART manifest for a single userapp.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/{PHA_EMAIL}/manifest for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.app_record_list(pha_email=None, body={}, headers={})

Return a list of all records that have this pha enabled.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/(PHA_EMAIL)/records/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.autonomous_access_token(record_id=None, pha_email=None, body={}, headers={}, content_type=None)

Fetch an access token for an autonomous app to access a record.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /apps/(PHA_EMAIL)/records/(RECORD_ID)/access_token for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.smart_capabilities()

SMART Capabilities

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /capabilities/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenets_delete(carenets_id=None, headers={})

Delete a carenet.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /carenets/(CARENET_ID) for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenets_account_list(carenets_id=None, body={}, headers={})

List the accounts in a carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/(CARENET_ID)/accounts/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.carenet_account_create(carenet_id=None, body={}, headers={}, content_type=None)

Add an account to a carenet.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /carenets/{CARENET_ID}/accounts/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_account_delete(account_id=None, carenet_id=None, headers={})

Remove an account from a carenet.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_account_permissions(account_id=None, carenet_id=None, body={}, headers={})

List the permissions of an account within a carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID}/permissions for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_apps_list(carenet_id=None, body={}, headers={})

List Apps within a given carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/apps/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_apps_delete(pha_email=None, carenet_id=None, headers={})

Remove an app from a given carenet.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /carenets/{CARENET_ID}/apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.carenet_apps_create(pha_email=None, carenet_id=None, body={}, headers={}, content_type=None)

Add an app to a carenet

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See `PUT /carenets/{CARENET_ID}/apps/{PHA_EMAIL}` for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_app_permissions(pha_email=None, carenet_id=None, body={}, headers={})

Retrieve the permissions for an app within a carenet. NOT IMPLEMENTED.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See `GET /carenets/{CARENET_ID}/apps/{PHA_EMAIL}/permissions` for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

IndivoClient.read_demographics_carenet(carenet_id=None, body={}, headers={})

Read demographics from a carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See `GET /carenets/{CARENET_ID}/demographics` for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

IndivoClient.carenets_document_list(carenet_id=None, body={}, headers={})

List documents from a given carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See `GET /carenets/{CARENET_ID}/documents/` for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

IndivoClient.carenets_document(document_id=None, carenet_id=None, body={}, headers={})

Return a document from a carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See `GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID}` for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.
IndivoClient.carenets_document_meta(document_id=None, carenet_id=None, body={}, headers={})

Fetch the metadata of a record-specific document via a carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenets_record(carenet_id=None, body={}, headers={})

Get basic information about the record to which a carenet belongs.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/record for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenets_rename(carenet_id=None, body={}, headers={}, content_type=None)

Change a carenet’s name.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /carenets/{CARENET_ID}/rename for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenets_equipment_list(carenet_id=None, body={}, headers={})

List the equipment data for a given carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/reports/minimal/equipment/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenets_measurement_list(carenet_id=None, lab_code=None, body={}, headers={})

List the measurement data for a given carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/reports/minimal/measurements/{LAB_CODE}/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.carenet_procedure_list(carenet_id=None, body={}, headers={})
List the procedure data for a given carenet.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/reports/minimal/procedures/ for valid parameters and the returned content.

This call will return a tuple (response, content), where `response` is an instance of httplib2.Response, and `content` is the raw content returned by Indivo.

IndivoClient.carenet_simple_clinical_notes_list(carenet_id=None, body={}, headers={})
List the simple_clinical_notes data for a given carenet.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/reports/minimal/simple-clinical-notes/ for valid parameters and the returned content.

This call will return a tuple (response, content), where `response` is an instance of httplib2.Response, and `content` is the raw content returned by Indivo.

IndivoClient.carenet_generic_list(data_model=None, carenet_id=None, body={}, headers={})
List the Model data for a given carenet.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/reports/{DATA_MODEL}/ for valid parameters and the returned content.

This call will return a tuple (response, content), where `response` is an instance of httplib2.Response, and `content` is the raw content returned by Indivo.

IndivoClient.coding_systems_list()
List available coding systems. NOT IMPLEMENTED.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /codes/systems/ for valid parameters and the returned content.

This call will return a tuple (response, content), where `response` is an instance of httplib2.Response, and `content` is the raw content returned by Indivo.

IndivoClient.coding_system_query(system_short_name=None, body={}, headers={})
Query a coding system for a value.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /codes/systems/{SYSTEM_SHORT_NAME}/query for valid parameters and the returned content.

This call will return a tuple (response, content), where `response` is an instance of httplib2.Response, and `content` is the raw content returned by Indivo.
IndivoClient.exchange_token(body={}, headers={}, content_type=None)

Exchange a request token for a valid access token.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /oauth/access_token for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.request_token_approve(reqtoken_id=None, body={}, headers={}, content_type=None)

Indicate a user’s consent to bind an app to a record or carenet.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /oauth/internal/request_tokens/{REQTOKEN_ID}/approve for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.request_token_claim(reqtoken_id=None, body={}, headers={}, content_type=None)

Claim a request token on behalf of an account.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /oauth/internal/request_tokens/{REQTOKEN_ID}/claim for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.request_token_info(reqtoken_id=None, body={}, headers={})

Get information about a request token.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /oauth/internal/request_tokens/{REQTOKEN_ID}/info for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.session_create(body={}, headers={}, content_type=None)

Authenticate a user and register a web session for them.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /oauth/internal/session_create for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.surl_verify([body={}, headers={}])

Verify a signed URL.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /oauth/internal/surl-verify for valid parameters and the returned content.

This call will return a tuple (*response, content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.request_token([body={}, headers={}, content_type=None])

Get a new request token, bound to a record or carenet if desired.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /oauth/request_token for valid parameters and the returned content.

This call will return a tuple (*response, content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.smart_ontology([body={}, headers={}])

Fetch the SMART ontology as RDF/XML.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /ontology for valid parameters and the returned content.

This call will return a tuple (*response, content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.record_create([body={}, headers={}, content_type=None])

Create a new record.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/ for valid parameters and the returned content.

This call will return a tuple (*response, content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.record_create_ext(principal_email=None, external_id=None, [body={}, headers={}, content_type=None])

Create a new record with an associated external id.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/external/{PRINCIPAL_EMAIL}/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (*response, content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
**IndivoClient.record_search** ([`body={}`, `headers={}`])

Search for records by label (usually the same as full name).

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/search for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

**IndivoClient.record** (`record_id=None`, `body={}`, `headers={}`)

Get information about an individual record.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID} for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

**IndivoClient.smart_allergies** (`record_id=None`, `body={}`, `headers={}`)

SMART allergy list, serialized as RDF/XML.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/allergies/ for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

**IndivoClient.smart_allergies_instance** (`record_id=None`, `model_id=None`, `body={}`, `headers={}`)

Retrieve a specific instance of a SMART Allergy/AllergyExclusion.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/allergies/{MODEL_ID} for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

**IndivoClient.record_phas** (`record_id=None`, `body={}`, `headers={}`)

List userapps bound to a given record.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/apps/ for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.
**IndivoClient.pha_record_delete**(record_id=None, pha_email=None[, headers={}])

Remove a userapp from a record.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /records/{RECORD_ID}/apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

**IndivoClient.record_pha**(record_id=None, pha_email=None[, body={}, headers={}])

Get information about a given userapp bound to a record.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

**IndivoClient.record_pha_enable**(record_id=None, pha_email=None[, body={}, headers={}, content_type=None])

Enable a userapp for a record.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/{RECORD_ID}/apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

**IndivoClient.record_app_document_list**(record_id=None, pha_email=None[, body={}, headers={}])

List record-app-specific documents.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

**IndivoClient.record_app_document_create**(record_id=None, pha_email=None[, body={}, headers={}, content_type=None])

Create a record-app-specific Indivo document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.record_app_document_create_or_update_ext(record_id=None, external_id=None, pha_email=None, body=[], headers={}, content_type=None)

Create or Overwrite a record-app-specific Indivo document with an associated external id.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.record_app_document_create_or_update_ext(record_id=None, external_id=None, pha_email=None, body=[], headers={}, content_type=None)

Create or Overwrite a record-app-specific Indivo document with an associated external id.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.record_app_document_meta_ext(record_id=None, external_id=None, pha_email=None, body=[], headers={})

Fetch the metadata of a record-app-specific document identified by external id.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.record_app_document_delete(record_id=None, pha_email=None, document_id=None, headers={})

Delete a record-app-specific document.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient\texttt{.record_app_specific_document}(record_id=None, pha_email=None, document_id=None, body={}, headers={})

Retrieve a record-app-specific document. 

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See \texttt{GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}} for valid parameters and the returned content.

This call will return a tuple \texttt{(response, content)}, where \texttt{response} is an instance of \texttt{httplib2.Response}, and \texttt{content} is the raw content returned by Indivo.

IndivoClient\texttt{.record_app_document_label}(record_id=None, pha_email=None, document_id=None, body={}, headers={}, content_type=None)

Set the label of a record-app-specific document.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See \texttt{PUT /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/label} for valid parameters and the returned content.

This call will return a tuple \texttt{(response, content)}, where \texttt{response} is an instance of \texttt{httplib2.Response}, and \texttt{content} is the raw content returned by Indivo.

IndivoClient\texttt{.record_app_document_meta}(record_id=None, pha_email=None, document_id=None, body={}, headers={})

Fetch the metadata of a record-app-specific document.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See \texttt{GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/meta} for valid parameters and the returned content.

This call will return a tuple \texttt{(response, content)}, where \texttt{response} is an instance of \texttt{httplib2.Response}, and \texttt{content} is the raw content returned by Indivo.

IndivoClient\texttt{.record_pha_setup}(record_id=None, pha_email=None, body={}, headers={}, content_type=None)

Bind an app to a record without user authorization.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See \texttt{POST /records/{RECORD_ID}/apps/{PHA_EMAIL}/setup} for valid parameters and the returned content.

This call will return a tuple \texttt{(response, content)}, where \texttt{response} is an instance of \texttt{httplib2.Response}, and \texttt{content} is the raw content returned by Indivo.

IndivoClient\texttt{.audit_record_view}(record_id=None, body={}, headers={})

Return audits of calls touching record.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See \texttt{GET /records/{RECORD_ID}/audits/} for valid parameters and the returned content.
This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.

```
IndivoClient.audit_document_view(record_id=None, document_id=None, body={}, headers={})
```

Return audits of calls touching \(\text{record}\) and \(\text{document}\).

\(\text{body}\) should contain query parameters for the call. \(\text{headers}\) should contain any additional HTTP headers to pass in with the call. See GET \(\text{/records/\{RECORD_ID\}/audits/documents/\{DOCUMENT_ID\}/}\) for valid parameters and the returned content.

This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.

```
IndivoClient.audit_function_view(record_id=None, document_id=None, function_name=None, body={}, headers={})
```

Return audits of calls to \(\text{function}\) touching \(\text{record}\) and \(\text{document}\).

\(\text{body}\) should contain query parameters for the call. \(\text{headers}\) should contain any additional HTTP headers to pass in with the call. See GET \(\text{/records/\{RECORD_ID\}/audits/documents/\{DOCUMENT_ID\}/functions/\{FUNCTION_NAME\}/}\) for valid parameters and the returned content.

This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.

```
IndivoClient.audit_query(record_id=None, body={}, headers={})
```

Select Audit Objects via the Query API Interface.

\(\text{body}\) should contain query parameters for the call. \(\text{headers}\) should contain any additional HTTP headers to pass in with the call. See GET \(\text{/records/\{RECORD_ID\}/audits/query/}\) for valid parameters and the returned content.

This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.

```
IndivoClient.autoshare_list(record_id=None, body={}, headers={})
```

For a single record, list all carenets that a given doctype is autoshared with.

\(\text{body}\) should contain query parameters for the call. \(\text{headers}\) should contain any additional HTTP headers to pass in with the call. See GET \(\text{/records/\{RECORD_ID\}/autoshare/bytype/}\) for valid parameters and the returned content.

This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.

```
IndivoClient.autoshare_list_bytype_all(record_id=None, body={}, headers={})
```

For a single record, list all doctypes autoshared into carenets.

\(\text{body}\) should contain query parameters for the call. \(\text{headers}\) should contain any additional HTTP headers to pass in with the call. See GET \(\text{/records/\{RECORD_ID\}/autoshare/bytype/all}\) for valid parameters and the returned content.

This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.
IndivoClient.autoshare_create(record_id=None, carenet_id=None, body={}, headers={}, content_type=None)

Automatically share all documents of a certain type into a carenet.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/set for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.autoshare_delete(record_id=None, carenet_id=None, body={}, headers={}, content_type=None)

Remove an autoshare from a carenet.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/unset for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_list(record_id=None, body={}, headers={})

List all carenets for a record.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/carenets/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_create(record_id=None, body={}, headers={}, content_type=None)

Create a new carenet for a record.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/carenets/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.read_demographics(record_id=None, body={}, headers={})

Read demographics from a record.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/demographics for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.set_demographics(record_id=None, body={}, headers={}, content_type=None)

Create or update demographics on a record.

- **body** should contain the data to send with the call.
- **headers** should contain any additional HTTP headers to pass in with the call.
- **content_type** should contain the MIME type of the data sent in the **body** argument. See PUT /records/{RECORD_ID}/demographics for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where **response** is an instance of httplib2.Response, and **content** is the raw content returned by Indivo.

IndivoClient.documents_delete(record_id=None, headers={})

Delete all documents associated with a record.

- **headers** should contain any additional HTTP headers to pass in with the call. See DELETE /records/{RECORD_ID}/documents/ for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where **response** is an instance of httplib2.Response, and **content** is the raw content returned by Indivo.

IndivoClient.record_document_list(record_id=None, body={}, headers={})

List record-specific documents.

- **body** should contain query parameters for the call.
- **headers** should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/ for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where **response** is an instance of httplib2.Response, and **content** is the raw content returned by Indivo.

IndivoClient.document_create(record_id=None, body={}, headers={}, content_type=None)

Create a record-specific Indivo Document.

- **body** should contain the data to send with the call.
- **headers** should contain any additional HTTP headers to pass in with the call.
- **content_type** should contain the MIME type of the data sent in the **body** argument. See POST /records/{RECORD_ID}/documents/ for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where **response** is an instance of httplib2.Response, and **content** is the raw content returned by Indivo.

IndivoClient.document_create_by_ext_id(record_id=None, external_id=None, pha_email=None, body={}, headers={}, content_type=None)

Create a record-specific Indivo Document with an associated external id.

- **body** should contain the data to send with the call.
- **headers** should contain any additional HTTP headers to pass in with the call.
- **content_type** should contain the MIME type of the data sent in the **body** argument. See PUT /records/{RECORD_ID}/documents/external/{PHA_EMAIL}/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where **response** is an instance of httplib2.Response, and **content** is the raw content returned by Indivo.
IndivoClient.record_document_label_ext

Set the label of a record-specific document, specified by external id.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /records/{RECORD_ID}/documents/external/{PHA_EMAIL}/{EXTERNAL_ID}/label for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_document_meta_ext

Fetch the metadata of a record-specific document identified by external id.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/external/{PHA_EMAIL}/{EXTERNAL_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.document_rels

Create a new relationship between two existing documents.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID_0}/rels/{REL}/{DOCUMENT_ID_1} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_specific_document

Retrieve a record-specific document.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/{DOCUMENT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.document_carenets(record_id=None, document_id=None, body={}, headers={})
List all the carenets into which a document has been shared.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/(RECORD_ID)/documents/(DOCUMENT_ID)/carenets/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.carenet_document_delete(record_id=None, carenet_id=None, document_id=None, headers={})
Unshare a document from a given carenet.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /records/(RECORD_ID)/documents/(DOCUMENT_ID)/carenets/(CARENET_ID) for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.carenet_document_placement(record_id=None, carenet_id=None, document_id=None, body={}, headers={}, content_type=None)
Place a document into a given carenet.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/(RECORD_ID)/documents/(DOCUMENT_ID)/carenets/(CARENET_ID) for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.autoshare_revert(record_id=None, carenet_id=None, document_id=None, body={}, headers={}, content_type=None)
Revert the document-sharing of a document in a carent to whatever rules are specified by autoshares. NOT IMPLEMENTED.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/(RECORD_ID)/documents/(DOCUMENT_ID)/carenets/(CARENET_ID)/autoshare-revert for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_document_label(record_id=None, document_id=None, body={}, headers={}, content_type=None)
Set the label of a record-specific document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See
PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/label for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_document_meta(record_id=None, document_id=None, body={}, headers={})

Fetch the metadata of a record-specific document.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.update_document_meta(record_id=None, document_id=None, body={}, headers={}, content_type=None)

Set metadata fields on a document. NOT IMPLEMENTED.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.document_remove_nevershare(record_id=None, document_id=None, headers={})

Remove the nevershare flag from a document.

headers should contain any additional HTTP headers to pass in with the call. See DELETE /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.document_set_nevershare(record_id=None, document_id=None, body={}, headers={}, content_type=None)

Flag a document to never be shared, anywhere.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.get_documents_by_rel(record_id=None, rel=None, document_id=None, body={}, headers={})

Get all documents related to the passed document_id by a relation of the passed relation-type.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL}/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.document_create_by_rel(record_id=None, rel=None, document_id=None, body={}, headers={}, content_type=None)

Create a document and relate it to an existing document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL}/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.document_create_by_rel_with_ext_id(record_id=None, rel=None, external_id=None, pha_email=None, document_id=None, body={}, headers={}, content_type=None)

Create a document, assign it an external id, and relate it to an existing document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL}/external/{PHA_EMAIL}/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.document_version(record_id=None, document_id=None[, body={}, headers={}, content_type=None])

Create a new version of a record-specific document.

`body` should contain the data to send with the call. `headers` should contain any additional HTTP headers to pass in with the call. `content_type` should contain the MIME type of the data sent in the `body` argument. See POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

IndivoClient.document_version_by_ext_id(record_id=None, external_id=None, pha_email=None, document_id=None[, body={}, headers={}, content_type=None])

Create a new version of a record-specific document and assign it an external id.

`body` should contain the data to send with the call. `headers` should contain any additional HTTP headers to pass in with the call. `content_type` should contain the MIME type of the data sent in the `body` argument. See PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace/external/{PHA_EMAIL}/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

IndivoClient.document_set_status(record_id=None, document_id=None[, body={}, headers={}, content_type=None])

Set the status of a record-specific document.

`body` should contain the data to send with the call. `headers` should contain any additional HTTP headers to pass in with the call. `content_type` should contain the MIME type of the data sent in the `body` argument. See POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/set-status for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

IndivoClient.document_status_history(record_id=None, document_id=None[, body={}, headers={}])

List all changes to a document’s status over time.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/status-history for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

IndivoClient.document_versions(record_id=None, document_id=None[, body={}, headers={}])

Retrieve the versions of a document.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/versions/ for valid parameters and the returned content.
This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_send_message(record_id=None, message_id=None, body={}, headers={}, content_type=None)

Send a message to a record.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /records/{RECORD_ID}/inbox/{MESSAGE_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_message_attach(record_id=None, attachment_num=None, message_id=None, body={}, headers={}, content_type=None)

Attach a document to an Indivo message.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_notify(record_id=None, body={}, headers={}, content_type=None)

Send a notification about a record to all accounts authorized to be notified.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /records/{RECORD_ID}/notify for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_get_owner(record_id=None, body={}, headers={})

Get the owner of a record.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/owner for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.record_set_owner(record_id=None, body={}, headers={}, content_type=None)

Set the owner of a record.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /records/{RECORD_ID}/owner for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.report_ccr(record_id=None, body={}, headers={})

Export patient data as a Continuity of Care Record (CCR) document.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/experimental/ccr for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.equipment_list(record_id=None, body={}, headers={})

List the equipment data for a given record.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/minimal/equipment/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.measurement_list(record_id=None, lab_code=None, body={}, headers={})

List the measurement data for a given record.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/minimal/measurements/{LAB_CODE}/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.procedure_list(record_id=None, body={}, headers={})

List the procedure data for a given record.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/minimal/procedures/ for valid parameters and the returned content.

This call will return a tuple (response, content), where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

IndivoClient.simple_clinical_notes_list(record_id=None, body={}, headers={})

List the simple_clinical_notes data for a given record.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/minimal/simple-clinical-notes/ for valid parameters and the returned content.

This call will return a tuple (response, content), where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

IndivoClient.generic_list(record_id=None, data_model=None, body={}, headers={})

List the Model data for a given record.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/{DATA_MODEL}/ for valid parameters and the returned content.

This call will return a tuple (response, content), where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

IndivoClient.record_shares(record_id=None, body={}, headers={})

List the shares of a record.

`body` should contain query parameters for the call. `headers` should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/shares/ for valid parameters and the returned content.

This call will return a tuple (response, content), where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

IndivoClient.record_share_add(record_id=None, body={}, headers={}, content_type=None)

Fully share a record with another account.

`body` should contain the data to send with the call. `headers` should contain any additional HTTP headers to pass in with the call. `content_type` should contain the MIME type of the data sent in the `body` argument. See POST /records/{RECORD_ID}/shares/ for valid parameters and the returned content.

This call will return a tuple (response, content), where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.
**IndivoClient.record_share_delete** *(record_id=None, other_account_id=None[, headers={}])*

Undo a full record share with an account.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /records/{RECORD_ID}/shares/{OTHER_ACCOUNT_ID} for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

**IndivoClient.record_share_delete** *(record_id=None, other_account_id=None[, body={}, headers={}, content_type=None])*

Undo a full record share with an account.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/shares/{OTHER_ACCOUNT_ID}/delete for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

**IndivoClient.smart_generic** *(record_id=None, model_name=None[, body={}, headers={}])*

SMART-compatible alias for the generic list view: returns data_models serialized as SMART RDF.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/{MODEL_NAME}/ for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

**IndivoClient.smart_generic_instance** *(record_id=None, model_id=None, model_name=None[, body={}, headers={}])*

Retrieve a specific instance of a SMART model.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/{MODEL_NAME}/{MODEL_ID} for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

**IndivoClient.get_version** *(body={}, headers={})*

Return the current version of Indivo.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /version for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
3.15 Registering Apps with Indivo

As of version 2.0, Indivo has an official process for adding and managing apps on a given instance. For those of you familiar with using `indivo_data.xml` for adding apps, the `<machine_apps>` and `<user_apps>` tags will no longer be respected, and will not add apps to Indivo.

3.15.1 App Manifests

Previous releases of Indivo have relied on an implicit XML syntax in `indivo_data.xml` for representing an app within Indivo. Now, applications must provide a declared manifest describing the application and its requirements. We have adopted the SMART Project’s syntax for describing apps in manifests, but have added some Indivo-specific parameters to accommodate the way Indivo represents applications. For compatibility, any valid SMART manifest will also be a valid Indivo manifest.

See The SMART Project’s documentation for a description of the basic syntax of a manifest, which is JSON based. Below, we describe only the Indivo-specific modifications.

User Apps

New Fields

An Indivo user-app may define (beyond the SMART-supported manifest fields) any of the following additional properties in its manifest:

- `oauth_callback_url`: A callback URL for Indivo-style OAuth access
- `autonomous_reason`: An explanation for why the app requires offline access to patient records
- `has_ui`: `true` or `false`, whether the app can be displayed in a browser.
- `frameable`: `true` or `false`, whether the app should be loaded in an iframe in the Indivo UI.
- `indivo_version`: Required version of Indivo for compatibility

Changes from the `indivo_data.xml` fields

The following fields have changed names to match the SMART manifest fields:

- `email`: Has been renamed `id`, per SMART.
- `start_url_template`: Has been renamed `index`. If using Indivo-style OAuth authentication, the same templating parameters may be passed in the URL (i.e. `{record_id}`)
- `is_autonomous`: Has been moved to the SMART `mode` property. Acceptable modes are:
  - `background`: This app will act like an autonomous Indivo app.
  - `ui`: This app will act like a non-autonomous Indivo app.

Example

```json
{
    "name" : "Problems",
    "description" : "Display a list of problems, or enter new ones."
    "author" : "Arjun Sanyal, Children’s Hospital Boston"
}
```
UI and Admin Apps

New Fields

An Indivo user-app may define (beyond the SMART-supported manifest fields) any of the following additional properties in its manifest:

- **ui_app**: true or false. Whether the machineapp is a UIApp ('chrome app').
- **indivo_version**: Required version of Indivo for compatibility

Changes from the indivo_data.xml fields

The following fields have changed names to match the SMART manifest fields:

- **email**: Has been renamed **id**, per SMART.
- **app_type**: Please use the **ui_app** field, above.

Example

```json
{
    "name": "Sample UI App",
    "description": "The reference Indivo UI App",
    "author": "Ben Adida, Travers Franckle, Arjun Sanyal, Pascal Pfiffner, Daniel Haas. Children’s Hospital Boston",
    "id": "chrome@apps.indivo.org",
    "version": "2.0.0",
    "indivo_version": "2.0.0",
    "ui_app": true
}
```

3.15.2 App oAuth Credentials

When authenticating to Indivo using traditional oAuth, applications must provided Indivo with their consumer key and a shared consumer_secret. As this secret is private and should not be shared with other apps (i.e., via a call to GET /apps/), it should be registered in a separate file. We therefore define a simple JSON format for specifying app oAuth credentials, which is simple JSON and has two fields:

- **consumer_key**: The OAuth consumer key for the app.
• **consumer_secret**: The OAuth consumer secret for the app.

Here’s a sample credentials file, for our built-in Problems app:

```json
{
    "consumer_key": "problems@apps.indivo.org",
    "consumer_secret": "SECRETFORTHEPROBLEMSAPP:CHANGEME"
}
```

**Notes:**

- If your app is a SMART app, you probably haven’t explicitly generated a ‘consumer key’. You should set the `consumer_key` field of the credentials file to match the `id` field of your app manifest file.

- If your app is a SMART CONNECT app (or connects to Indivo using *In-Browser Connect Authentication*), you do not need a consumer secret. In such a case, set the `consumer_secret` field of the credentials file to the empty string: `''`.

### 3.15.3 Managing the Registered Apps

Thus, to register an app with Indivo, you need two files: an app manifest (`manifest.json`) and a credentials file (`credentials.json`).

Changing the set of registered apps in Indivo is now drag-and-drop, as with our process for managing datatypes and schemas. To add, remove, or change an app, you’ll need to:

- Create a manifest and credentials file for the app (or modify existing manifests/credentials)
- Drop the files into the filesystem
- Sync the database with the filesystem

### Apps in the Indivo Filesystem

Indivo apps currently have the following layout on the filesystem:

```
indivo_server/
    registered_apps/
        admin/
        ui/
        user/
            allergies/
                manifest.json
                credentials.json
            ...
```

To add an app to the filesystem, simply add a subdirectory under `indivo_server/registered_apps/admin`, `indivo_server/registered_apps/ui`, or `indivo_server/registered_apps/user` (depending on the type of your app), and drop a manifest and a credentials file into that directory.

To remove an app, just delete its directory.

To change an app’s manifest or credentials, just modify the appropriate `manifest.json` or `credentials.json` file.

### Syncing the Database with the Filesystem

To alert Indivo that you’ve changed the registered apps, run (from `indivo_server/`):
This will process the list of registered apps and sync any additions, deletes or updates to the database.

**Resetting Indivo**

With the new system, there is **NO NEED TO RESET INDIVO TO ADD APPS!** Simply run the `sync_apps` command, above.

When you do reset Indivo, the reset script now calls `sync_apps`, which will add all of the registered apps to Indivo.

**3.16 Sample Data in Indivo**

Testing out new Indivo apps is difficult at best with no patient data to run against. That’s why we’ve added the capability to create records with sample patient data pre-loaded into them. There are two pathways for getting sample data into Indivo: using the `indivo_data.xml` file and the `reset.py` script, and putting Indivo into Demo Mode.

**3.16.1 Using `indivo_data.xml`**

The simplest way to get sample data into Indivo is to use the basic reset script. When you’re setting up `indivo_data.xml`, just add a `data_profile` attribute to any `<record>` tags that you’re creating. The default `indivo_data.xml` file has an example of this already: it loads the data profile for `patient_2` into the John S. Smith record.

As described below, you can determine available data profiles by looking at the subdirectories of `settings.SAMPLE_DATA_DIR`.

**3.16.2 Using Demo Mode**

If you would like to run Indivo fully populated by sample data (as we do on our developer’s sandbox), you can put Indivo into **Demo Mode**. In this mode, all newly created accounts are immediately set up with records pre-loaded with sample data.

You can do this by configuring the following settings in `settings.py`:

- **SAMPLE_DATA_DIR** The directory where sample data is located.
- **DEMO_MODE** Puts Indivo into Demo Mode if set to True.
- **DEMO_PROFILES** A dictionary mapping record labels to data profiles to load for each new account. For example, if the value of this settings were:

```python
{'John Doe': 'patient_1',
 'Robert Frost': 'bob',
 'Ted Kennedy': 'patient_2',
}
```

Then for each new account, three new records would be created. The first would have a label of ‘John Doe’, and be populated by the data profile ‘patient_1’. The second would have a label of ‘Robert Frost’, and be populated by the data profile ‘bob’. The third would have a label of ‘Ted Kennedy’, and be populated by the data profile ‘patient_2’.
Note: Demo Mode autocreates records any time the API call to create an account, POST /accounts/, is called. This means that any records created through other means (i.e. by a call to POST /records/) will not be populated with data. If your registration UI or admin app handles record creation, this could lead to the existence of some records populated with sample data, and others without it.

3.16.3 Available Sample Data

Any data in the directory specified by settings.SAMPLE_DATA_DIR (settings.APP_HOME/sample_data by default) is available for loading into Indivo. Data in SAMPLE_DATA_DIR should look like:

```
profile_1/  # Data profiles. Each directory should correspond to a single patient.
bob/       # For example, this is the data profile that can be referenced as 'bob'
    ...
profile_n/
    Demographics.xml # An optional demographics document.
    
    doc_1.xml # XML Data to load goes here.
    doc_2.xml # File names MUST be prefixed with 'doc_'.
    ...
    doc_n.xml
    
    doc_1.pdf # Other extensions are treated as binary docs.
    doc_2.pdf # Also prefix names with 'doc_'.
    ...
```

Namely, the data directory should have multiple subdirectories, each representing one patient’s data. Within a patient’s directory, there might be a Demographics.xml file. There will also be any number of data files, labeled doc_(NAME).(EXTENSION), where NAME can be anything, and EXTENSION describes the type of data in the file.

We’ve provided you with a few sample patients to get started with, but you should feel free to add data that is useful to your specific Indivo installation.

3.16.4 Adding To the Available Sample Data

Adding sample data to Indivo is trivial: simply add files to settings.SAMPLE_DATA_DIR, making sure to preserve the directory structure described above. You can either:

- Add data to an existing profile, by dropping new data files into that profile’s directory, or
- Add a new profile, by creating a new subdirectory of SAMPLE_DATA_DIR. Make sure to add a demographics document for the new profile.

3.17 Indivo Schemas

3.17.1 Introduction

Schemas in Indivo are used to describe valid formats in which data may enter Indivo. Right now, we use XSDs as schemas, since we accept input data only in XML form, but in the future we might extend this to include schemas for validating other formats of data (OWL for RDF, etc.). Note that schemas describe the format of input data only! For information on how data is processed and stored in Indivo, see Data Pipeline.
There are a number of XML standards for medical activities, ranging from the CCR summary to the highly detailed CCD. None of these are particularly well tuned to the needs of a PCHR, where an individual datum may come from a hospital data feed, or from patient-based data entry. The Indivo schemas are built to serve the specific PCHR needs. Importantly, the Indivo schemas use standard coding systems wherever possible. The schemas are also ready for new coding systems as they emerge, especially in the realm of personally-controlled medicine with simplified terminology.

3.17.2 What if I want to store data that doesn’t match an Indivo schema?

Indivo X is designed to accept documents that conform to any XML schema, such as CCR, and even documents that are not XML, i.e. PDFs, MPEG, etc....

XML documents that conform to the built-in schemas can be immediately transformed, via the Indivo X Data Pipeline, into individual datapoints, which can then be queried using the Indivo Reporting API. XML documents that conform to custom schemas are not processed, and therefore cannot be retrieved using the reporting API (though you can still access them with API calls for retrieving unprocessed documents, which will return them in their original XML form.

If you want to extend Indivo to enable querying over data input according to a new schema, see Adding Custom Schemas to Indivo.

3.17.3 Namespace and XML Types

All of the default Indivo X document schemas are in a single namespace:

http://indivo.org/vocab/xml/documents#

The use of the trailing # enables simple RDF-like concatenation of namespace prefix and suffix to generate a single type URL. For example, an SDMX document in the Indivo documents namespace will have as its type:

http://indivo.org/vocab/xml/documents#Models

3.17.4 Design Rationale for Inclusion vs. Relation

Indivo X brings the ability to relate documents to one another using metadata, rather than document payload. This is particularly important when the payload might not be under the user’s control, i.e. a CCR document. It can also be useful even in the design of new Indivo schemas.

One could imagine separation the prescription information from the medication information, having two documents related to one another rather than one bigger document. However, our design rationale for now is to keep medicatin and its prescription data in the same XML document because those two chunks of data are generated in the same event. If, at some point, Indivo stores prescription filling information, then it is likely that this information would be more appropriately stored in a separate, linked document.

3.17.5 Core Schemas

All schema files and sample instance documents are available at http://indivo.org/vocab/xml/. Note that these schemas are only the ones that come with Indivo by default. Each instance of Indivo might define additional, custom schemas that are not documented here. See Adding Custom Schemas to Indivo for instructions on how to add custom schemas to Indivo.
Metadata and Indivo Internal Data Structures

Indivo Document Metadata Schema

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- didn’t place this in the Indivo namespace because it’s not medical payload -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Principal">
    <xs:sequence>
      <xs:element name="fullname" type="xs:string" minOccurs="1" maxOccurs="1" />
      <!-- e.g. "fhh@apps.indivo.org" -->
      <xs:attribute name="id" type="xs:string" use="required" />
      <!-- e.g. "userapp" or "account" or "adminapp" -->
      <xs:attribute name="type" type="xs:string" use="required" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Relation">
    <xs:attribute name="type" type="xs:string" use="required" />
    <xs:attribute name="count" type="xs:integer" use="required" />
  </xs:complexType>
  <xs:element name="Document">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="createdAt" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
        <xs:element name="creator" type="Principal" minOccurs="1" maxOccurs="1" />
        <!-- if suppressedAt is non-null, then suppressor should be present -->
        <xs:element name="suppressedAt" type="xs:dateTime" minOccurs="0" maxOccurs="1" />
        <xs:element name="suppressor" type="Principal" minOccurs="0" maxOccurs="1" />
        <xs:element name="replacedBy" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="replaces" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="original" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="latest" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required" />
            <xs:attribute name="createdAt" type="xs:dateTime" use="required" />
            <xs:attribute name="createdBy" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

294 Chapter 3. App Developers
Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d5a484" size="77"
<createdAt>2009-05-04T17:05:33</createdAt>
<creator id="steve@indivo.org" type="account">
  <fullname>Steve Zabak</fullname>
</creator>
<suppressedAt>2009-05-06T17:05:33</suppressedAt>
<suppressor id="steve@indivo.org" type="account">
  <fullname>Steve Zabak</fullname>
</suppressor>
<original id="14c81023-c84f-496d-8b8e-9438280441d3" />
<latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
  <label>HBA1C reading</label>
  <status>active</status>
  <nevershare>false</nevershare>
  <relatesTo>
    <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
  </relatesTo>
  <isRelatedFrom>
    <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
  </isRelatedFrom>
</latest>
</Document>
```
Indivo Account Schema

An Indivo Account represents a single user of the system, with their basic info and the ways in which they authenticate. It is separate from a record.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
    <element name="Account">
        <complexType>
            <sequence>
                <element name="secret" type="string" minOccurs="0" maxOccurs="1" />
                <element name="fullName" type="string" minOccurs="1" maxOccurs="1" />
                <element name="contactEmail" type="string" minOccurs="1" maxOccurs="1" />
                <element name="lastLoginAt" type="dateTime" minOccurs="1" maxOccurs="1" />
                <element name="totalLoginCount" type="integer" minOccurs="1" maxOccurs="1" />
                <element name="failedLoginCount" type="integer" minOccurs="1" maxOccurs="1" />
                <element name="state" type="string" minOccurs="1" maxOccurs="1" />
                <element name="lastStateChange" type="dateTime" minOccurs="1" maxOccurs="1" />
                <element name="authSystem" minOccurs="0" maxOccurs="unbounded">
                    <complexType>
                        <attribute name="name" type="string" use="required" />
                        <attribute name="username" type="string" use="required" />
                    </complexType>
                </element>
            </sequence>
            <attribute name="id" type="string" use="required" />
        </complexType>
    </element>
</schema>
```

Example:

```xml
<Account id="joeuser@indivo.example.org">
    <fullName>Joe User</fullName>
    <contactEmail>joeuser@gmail.com</contactEmail>
    <lastLoginAt>2010-05-04T15:34:23Z</lastLoginAt>
    <totalLoginCount>43</totalLoginCount>
    <failedLoginCount>0</failedLoginCount>
    <state>active</state>
    <lastStateChange>2009-04-03T13:12:12Z</lastStateChange>
    <authSystem name="password" username="joeuser" />
    <authSystem name="hospital_sso" username="Joe_User" />
</Account>
```

Indivo PHA Schema

Information describing a Personal Health App (User App). Can be wrapped into a set of Apps.

Schema:
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">

<x:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">

<x:complexType name="App">
  <xs:sequence>
    <xs:element name="startURLTemplate" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1" />
    <xs:element name="description" type="xs:string" minOccurs="1" maxOccurs="1" />
    <xs:element name="autonomous" type="xs:boolean" minOccurs="1" maxOccurs="1" />
    <xs:element name="autonomousReason" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="frameable" type="xs:boolean" minOccurs="1" maxOccurs="1" />
    <xs:element name="ui" type="xs:boolean" minOccurs="1" maxOccurs="1" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType>

<x:element name="App" type="App" />

<x:element name="Apps">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="App" type="App" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>

Example:

<App id="problems@apps.indivo.org">
  <startURLTemplate>http://problems.indivo.org/auth/start?record_id={record_id}&amp;carenet_id={carenet_id}</startURLTemplate>
  <name>Problem List</name>
  <description>Managing your problem list</description>
  <autonomous>false</autonomous>
  <frameable>true</frameable>
  <ui>true</ui>
</App>

Example of multiple apps:

<Apps>
  <App id="problems@apps.indivo.org">
    <startURLTemplate>http://problems.indivo.org/auth/start?record_id={record_id}&amp;carenet_id={carenet_id}</startURLTemplate>
    <name>Problem List</name>
    <description>Managing your problem list</description>
    <autonomous>false</autonomous>
    <frameable>true</frameable>
    <ui>true</ui>
  </App>
</Apps>

Indivo Audit Log Schema

As of Beta 3, the logs will be returned as Indivo Reports according to the Indivo Reporting Schema. Each report item will be of type <AuditEntry>, as defined below:
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="AuditEntry">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="BasicInfo" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="datetime" type="xs:dateTime" use="required" />
            <xs:attribute name="view_func" type="xs:string" use="required" />
            <xs:attribute name="request_successful" type="xs:boolean" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="PrincipalInfo" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="effective_principal" type="xs:string" use="required" />
            <xs:attribute name="proxied_principal" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="Resources" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="carenet_id" type="xs:string" use="required" />
            <xs:attribute name="record_id" type="xs:string" use="required" />
            <xs:attribute name="pha_id" type="xs:string" use="required" />
            <xs:attribute name="document_id" type="xs:string" use="required" />
            <xs:attribute name="external_id" type="xs:string" use="required" />
            <xs:attribute name="message_id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="RequestInfo" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="req_url" type="xs:string" use="required" />
            <xs:attribute name="req_ip_address" type="xs:string" use="required" />
            <xs:attribute name="req_domain" type="xs:string" use="required" />
            <xs:attribute name="req_method" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="ResponseInfo" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="resp_code" type="xs:integer" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<AuditEntry>
  <BasicInfo datetime="2011-04-27T17:32:23Z" view_func="get_document" request_successful="true" />
  <PrincipalInfo effective_principal="myapp@apps.indivohealth.org" proxied_principal="me@indivohealth.org" />
  <Resources carenet_id="" record_id="123" pha_id="" document_id="acd" external_id="" message_id="" />
  <RequestInfo req_url="/records/123/documents/acd/" req_ip_address="127.0.0.1" req_domain="localhost" />
  <ResponseInfo resp_code="200" />
</AuditEntry>
```
Indivo Carenet Schema

A list of carenets is returned when a user/app wants to know how a document is shared. However, this same list of carenets might be used in a different setting. Thus, the “mode” attribute is optional. It indicates whether sharing in this carenet was done explicitly, or via some implicit auto-share rule.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="Carenets">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Carenet" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="mode" type="xs:string" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="record_id" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```xml
<Carenets record_id="123">
  <Carenet id="789" name="Work/School" mode="explicit"/>
</Carenets>
```

Indivo Document Status History Schema

When a document’s status changes (archived, etc..), its history of changes is documented and available in this schema.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="DocumentStatusHistory">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DocumentStatus" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="reason" type="xs:string" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
            <xs:attribute name="by" type="xs:string" use="required"/>
            <xs:attribute name="at" type="xs:dateTime" use="required"/>
            <xs:attribute name="status" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="document_id" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Example:

```xml
<DocumentStatusHistory document_id="456">
  <DocumentStatus by="joeuser@indivo.example.org" at="2010-09-03T12:45:12Z" status="archived">
    <reason>no longer relevant</reason>
  </DocumentStatus>
</DocumentStatusHistory>
```

**Indivo Inbox Message Schema**

Indivo messages, sent to accounts (sometimes via a record), are represented with the following schema.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:complexType name="Message">
    <xs:sequence>
      <xs:element name="sender" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="received_at" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
      <xs:element name="read_at" type="xs:dateTime" minOccurs="0" maxOccurs="1" />
      <xs:element name="subject" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="severity" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="record" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
      <xs:element name="attachment" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="num" type="xs:integer" use="required" />
          <xs:attribute name="type" type="xs:string" use="required" />
          <xs:attribute name="size" type="xs:integer" use="required" />
          <xs:attribute name="doc_id" type="xs:string" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>
  <xs:element name="Message" type="Message" />
</xs:complexType>

<xs:element name="Messages">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Message" type="Message" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Example:
Another Example:

<Message id="879">
  <sender>doctor@example.indivo.org</sender>
  <received_at>2010-09-04T14:12:12Z</received_at>
  <subject>your test results are looking good</subject>
  <severity>normal</severity>
  <record id="123" />
  <attachment num="1" type="http://indivo.org/vocab/xml/documents#Lab" size="12546" />
</Message>

Indivo Notification Schema

The Indivo Healthfeed includes notifications, represented by this schema:

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="Notifications">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Notification">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="sender" type="xs:string" minOccurs="1" maxOccurs="1" />
              <xs:element name="received_at" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
              <xs:element name="content" type="xs:string" minOccurs="1" maxOccurs="1" />
              <xs:element name="record" minOccurs="1" maxOccurs="1">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:string" use="required" />  
                  <xs:attribute name="label" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
              <xs:element name="document" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:string" use="required" />
                  <xs:attribute name="label" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3.17. Indivo Schemas
Example:

```xml
<Notifications>
  <Notification id="468">
    <sender>labs@apps.indivo.org</sender>
    <received_at>2010-09-03T15:12:12Z</received_at>
    <content>A new lab result has been delivered to your account</content>
    <record id="123" label="Joe User" />
    <document id="579" label="Lab Test 2" />
  </Notification>
</Notifications>
```

**Indivo Permissions Schema**

Coming Soon...

**Indivo Record Schema**

The basic info for an Indivo record. Some of the attributes are there for indicating sharing relationships.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">

  <xs:element name="Record">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="demographics" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="document_id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="created" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="at" type="xs:dateTime" use="optional" />
            <xs:attribute name="by" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required" />
      <xs:attribute name="label" type="xs:string" use="required" />
      <xs:attribute name="shared" type="xs:boolean" use="optional" />
      <xs:attribute name="carenet_id" type="xs:string" use="optional" />
      <xs:attribute name="carenet_name" type="xs:string" use="optional" />
      <xs:attribute name="role_label" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Example:

```xml
<Record id="123" label="Joe User" shared="true">
    <demographics document_id="467" />
    <created at="2010-10-23T10:23:34Z" by="indivoconnector@apps.indivo.org" />
</Record>
```

**Indivo Request Token Schema**

The Indivo UI Server needs to manage request tokens for apps so that it can display the appropriate authorization screens. This schema makes use of the *Indivo PHA Schema*.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
    <xs:include schemaLocation="../pha/pha.xsd" />

    <xs:element name="RequestToken">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="record" minOccurs="1" maxOccurs="1">
                    <xs:complexType>
                        <xs:attribute name="id" type="xs:string" use="optional" />
                    </xs:complexType>
                </xs:element>

                <xs:element name="carenet" minOccurs="1" maxOccurs="1">
                    <xs:complexType>
                        <xs:attribute name="id" type="xs:string" use="optional" />
                    </xs:complexType>
                </xs:element>

                <xs:element name="kind" type="xs:string" minOccurs="1" maxOccurs="1" />

                <xs:element name="App" type="App" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
            <xs:attribute name="token" type="xs:string" use="required" />
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Example:

```xml
<RequestToken token="XYZ">
    <record id="123" />
    <carenet />
    <kind>new</kind>
    <App id="problems@apps.indivo.org">
        <name>Problem List</name>
        <description>Managing your list of problems</description>
        <autonomous>false</autonomous>
    </App>
</RequestToken>
```
Indivo Coded Values

A coded value is a value taken from a coding system. It consists of a reference to the coding system (a URL), the code value, and the human-readable string. When the coding system is not used but a manual value is entered, the coding system and coded value are absent, leaving only the human-readable string.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
  xmlns:indivo="http://indivo.org/vocab/xml/documents#" elementFormDefault="qualified">
  <xs:complexType name="CodedValue">
    <xs:complexContent>
      <xs:extension base="xs:string">
        <xs:attribute name="type" type="xs:anyURI" use="optional" />
        <xs:attribute name="value" type="xs:string" use="optional" />
        <xs:attribute name="abbrev" type="xs:string" use="optional" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

• When a document comes into Indivo, its coded values may be expanded (with abbreviation and element content) or not (just the code and coding system).

• We will encourage applications to provide expanded coded values, but this will not be required.

• Reports will provide abbreviations and full names for all relevant codes by looking up against Indivo-stored copies of the coding systems. Documents will not be modified from what the sources send us, to follow the principles of store exactly the original data source (that’s required because the documents might be digitally signed.)

• Reports can flag codes whose abbreviations and full names do not match the coding system data (but we always show by default what the document says, we trust the source, not the coding system.)

• We then need a code lookup API for viewing single documents.

• codes.indivo.org will provide an API for interpreting codes.

Indivo Data Values Schema

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
  <xs:complexType name="Range">
    <xs:sequence>
      <!-- a missing minimum means < max -->
      <xs:element name="minimum" type="xs:double" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
</xs:schema>
```
<!-- a missing maximum means > min -->
<xs:element name="maximum" type="xs:double" minOccurs="0" maxOccurs="1" />

<!-- technically this schema allows a range with neither min nor max, which doesn’t mean much, -->
<xs:element name="unit" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
</xs:sequence>
</xs:complexType>

<!-- an ordinal, i.e. "2+" is coded using textValue -->
<xs:complexType name="ValueAndUnit">
  <xs:sequence>
    <xs:element name="value" type="xs:double" minOccurs="0" maxOccurs="1" />
    <xs:element name="textValue" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="unit" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<!-- a result is an abstract base, only subtypes can be used -->
<xs:complexType name="Result" abstract="true">
  <xs:sequence>
    <!-- HL7 defines flag types -->
    <xs:element name="flag" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ResultInRange">
  <xs:complexContent>
    <xs:extension base="indivo:Result">
      <xs:sequence>
        <xs:element name="valueAndUnit" type="indivo:ValueAndUnit" minOccurs="1" maxOccurs="1" />
        <xs:element name="normalRange" type="indivo:Range" minOccurs="0" maxOccurs="1" />
        <!-- nontoxicrange as in, if it’s outside the range, it’s toxic -->
        <xs:element name="nonCriticalRange" type="indivo:Range" minOccurs="0" maxOccurs="1" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- untyped strings, e.g. "positive" -->
<xs:complexType name="ResultInSet">
  <xs:complexContent>
    <xs:extension base="indivo:Result">
      <xs:sequence>
        <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1" />
        <!-- the options should be listed in sensible order -->
        <xs:element name="option" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="normal" type="xs:boolean" use="required" />
                <xs:attribute name="description" type="xs:string" use="optional" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

3.17. Indivo Schemas 305
Indivo X Documentation, Release 2.0.0

<xs:complexType>
  <xs:element name="value" type="indivo:ValueAndUnit" />
  <xs:element name="range" type="indivo:Range" />
</xs:choice>
</xs:complexType>

<xs:complexType name="Concentration">
  <xs:complexContent>
    <xs:extension base="indivo:ValueOrRange" />
  </xs:complexContent>
</xs:complexType>

Indivo Provider Schema

A provider is typically an MD with an institution affiliation.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
  xmlns:indivo="http://indivo.org/vocab/xml/documents#"
  elementFormDefault="qualified">
  <complexType name="Provider">
    <sequence>
      <element name="name" type="string" minOccurs="1" maxOccurs="1" />
      <element name="institution" type="string" minOccurs="1" maxOccurs="1" />
    </sequence>
  </complexType>

  <complexType name="LabProvider">
    <sequence>
      <element name="name" type="string" minOccurs="1" maxOccurs="1" />
      <element name="address" type="string" minOccurs="1" maxOccurs="1" />
    </sequence>
  </complexType>

  <complexType name="Signature">
    <sequence>
      <element name="at" type="dateTime" minOccurs="0" maxOccurs="1" />
      <element name="provider" type="indivo:Provider" minOccurs="1" maxOccurs="1" />
    </sequence>
  </complexType>
</schema>
```
Reporting

Indivo Reporting Schema

See the schema for Indivo Document Metadata Schema and specific indivo:doc schemas.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
    elementFormDefault="qualified">

    <xs:complexType name="Report">
      <xs:sequence>
        <xs:element name="Meta" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Document" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Item" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:choice minOccurs="1" maxOccurs="1">
                <xs:element name="Allergy" />
                <xs:element name="Equipment" />
                <xs:element name="Immunization" />
                <xs:element name="LabReport" />
                <xs:element name="Medication" />
                <xs:element name="Problem" />
                <xs:element name="Procedure" />
                <xs:element name="SimpleClinicalNote" />
                <xs:element name="VitalSign" />
                <xs:element name="AggregateReport" />
              </xs:choice>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>

    <xs:element name="Reports">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Summary" minOccurs="1" maxOccurs="1">
            <xs:complexType>
              <xs:attribute name="total_document_count" use="optional" type="xs:int" />
              <xs:attribute name="limit" use="optional" type="xs:int" />
              <xs:attribute name="offset" use="optional" type="xs:int" />
              <xs:attribute name="order_by" use="optional" type="xs:string"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="QueryParams" minOccurs="1" maxOccurs="1">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="GroupBy" minOccurs="0" maxOccurs="1">
                  <xs:complexType>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

</xs:schema>
```
Indivo Aggregate Report Schema

This schema describes report items returned in aggregate form.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
    xmlns:indivo="http://indivo.org/vocab/xml/documents#"
    elementFormDefault="qualified">
```
<xs:complexType name="AggregateReport">
    <xs:attribute name="value" type="xs:string" use="required" />
    <xs:attribute name="group" type="xs:string" use="optional" />
</xs:complexType>

<xs:element name="AggregateReport" type="indivo:AggregateReport" />
</xs:schema>

Example:
<?xml version="1.0" encoding="utf-8" ?>

Indivo Generic Aggregate Reports Schema

This schema describes report items returned in XML aggregate form from Generic Reports.

Schema:
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:indivo="http://indivo.org/vocab/xml/documents#"
    elementFormDefault="qualified"
    targetNamespace="http://indivo.org/vocab/xml/documents#">

    <xs:complexType name="AggregateReport">
        <xs:attribute name="value" type="xs:string" use="required" />
        <xs:attribute name="group" type="xs:string" use="optional" />
    </xs:complexType>

    <xs:element name="AggregateReports">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="AggregateReport" type="AggregateReport" minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>

Example:
<?xml version="1.0" encoding="utf-8" ?>
<AggregateReports>
    <AggregateReport value="1" group="2009-07" />
    <AggregateReport value="4" group="2009-08" />
</AggregateReports>

Special Documents

Indivo Document Demographics Schema

Schema:
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:indivo="http://indivo.org/vocab/xml/documents#"
    elementFormDefault="qualified"
    targetNamespace="http://indivo.org/vocab/xml/documents#">
<xs:complexType name="Name">
  <xs:sequence>
    <xs:element name="familyName" type="xs:string" minOccurs="1" maxOccurs="1" />
    <xs:element name="givenName" type="xs:string" minOccurs="1" maxOccurs="1" />
    <xs:element name="middleName" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="prefix" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="suffix" type="xs:string" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="PhoneType">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="h"/>
    <xs:enumeration value="w"/>
    <xs:enumeration value="c"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="GenderType">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="female"/>
    <xs:enumeration value="male"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="Telephone">
  <xs:sequence>
    <xs:element name="type" type="indivo:PhoneType" minOccurs="0" maxOccurs="1" />
    <xs:element name="number" type="xs:string" minOccurs="1" maxOccurs="1" />
    <xs:element name="preferred" type="xs:boolean" minOccurs="0" maxOccurs="1" default="false" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Address">
  <xs:sequence>
    <xs:element name="country" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="city" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="postalCode" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="region" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="street" type="xs:string" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:element name="Demographics">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="dateOfBirth" type="xs:date" minOccurs="1" />
      <xs:element name="gender" type="indivo:GenderType" minOccurs="1" />
      <xs:element name="email" type="xs:string" minOccurs="0" />
      <xs:element name="ethnicity" type="xs:string" minOccurs="0" />
      <xs:element name="preferredLanguage" type="xs:string" minOccurs="0" />
      <xs:element name="race" type="xs:string" minOccurs="0" />
      <xs:element name="Name" type="indivo:Name" minOccurs="1" />
      <xs:element name="Telephone" type="indivo:Telephone" minOccurs="0" maxOccurs="2" />
      <xs:element name="Address" type="indivo:Address" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

3.17. Indivo Schemas 311
Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Demographics xmlns="http://indivo.org/vocab/xml/documents#">
  <dateOfBirth>1939-11-15</dateOfBirth>
  <gender>male</gender>
  <email>test@fake.org</email>
  <ethnicity>Scottish</ethnicity>
  <preferredLanguage>EN</preferredLanguage>
  <race>caucasian</race>
  <Name>
    <familyName>Wayne</familyName>
    <givenName>Bruce</givenName>
    <middleName>Quentin</middleName>
    <prefix>Mr</prefix>
    <suffix>Jr</suffix>
  </Name>
  <Telephone>
    <type>h</type>
    <number>555-5555</number>
    <preferred>true</preferred>
  </Telephone>
  <Telephone>
    <type>c</type>
    <number>555-6666</number>
  </Telephone>
  <Address>
    <country>USA</country>
    <city>Gotham</city>
    <postalCode>90210</postalCode>
    <region>secret</region>
    <street>1007 Mountain Drive</street>
  </Address>
</Demographics>
```

**Indivo Document Annotation Schema**

Medical documents can be annotated in Indivo, with a document that’s added in relation to the annotated document. This is a simple schema for these text-based annotations.

Schema:

Coming Soon!

Example:

Coming Soon!

The relationship to the annotated document is maintained in Indivo metadata, and the annotation can, optionally, store the SHA256 hash of the referenced document for robustness.
Medical Documents

Indivo Document Schema: Procedure

A procedure is effectively a surgical event (though some are not exactly surgical.)

See also the schema for Indivo Coded Values and for Indivo Provider Schema.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
    xmlns:indivo="http://indivo.org/vocab/xml/documents#" elementFormDefault="qualified">
    <element name="Procedure">
        <complexType>
            <sequence>
                <element name="datePerformed" type="dateTime" minOccurs="0" maxOccurs="1" />
                <element name="name" type="indivo:CodedValue" minOccurs="1" maxOccurs="1" />
                <element name="provider" type="indivo:Provider" minOccurs="0" />       
                <element name="location" type="string" minOccurs="0" maxOccurs="1" />
                <element name="comments" type="string" minOccurs="0" maxOccurs="1" />
            </sequence>
        </complexType>
    </element>
</schema>
```

Example:

```xml
<Procedure xmlns="http://indivo.org/vocab/xml/documents#">
    <datePerformed>2009-05-16T12:00:00</datePerformed>
    <name type="http://codes.indivo.org/procedures#" value="85" abbrev="append">Appendectomy</name>
    <provider>
        <name>Kenneth Mandl</name>
        <institution>Children's Hospital Boston</institution>
    </provider>
</Procedure>
```

Indivo Document Schema: Equipment

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
    xmlns:indivo="http://indivo.org/vocab/xml/documents#" elementFormDefault="qualified">
    <element name="Equipment">
        <complexType>
            <sequence>
                <element name="dateStarted" type="date" minOccurs="0" maxOccurs="1" />
                <element name="dateStopped" type="date" minOccurs="0" maxOccurs="1" />
                <element name="type" type="string" minOccurs="0" maxOccurs="1" />
                <element name="name" type="string" minOccurs="1" maxOccurs="1" />
            </sequence>
        </complexType>
    </element>
</schema>
```
<element name="vendor" type="string" minOccurs="0" maxOccurs="1" />
<element name="id" type="string" minOccurs="0" maxOccurs="1" />
<element name="description" type="string" minOccurs="0" maxOccurs="1" />
<element name="specification" type="string" minOccurs="0" maxOccurs="1" />
<element name="certification" type="string" minOccurs="0" maxOccurs="1" />
</sequence>
</complexType>
</element>
</schema>

Example:

<Equipment xmlns="http://indivo.org/vocab/xml/documents#">
  <dateStarted>2009-02-05</dateStarted>
  <dateStopped>2010-06-12</dateStopped>
  <type>cardiac</type>
  <name>Pacemaker</name>
  <vendor>Acme Medical Devices</vendor>
  <id>167-ABC-23</id>
  <description>it works</description>
  <specification>blah blah blah</specification>
</Equipment>

Indivo Document Schema: Simple Clinical Note

A full clinical note needs to contain a number of coded problems, etc. Some hospital systems do not have fully normalized clinical notes, in which case they can use this schema to store some simple attributes and the main free-form text of the note.

See also Indivo Coded Values and Indivo Provider Schema.

Schema:

<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://indivo.org/vocab/xml/documents#" elementFormDefault="qualified">
  <include schemaLocation="../../common/codes.xsd" />
  <include schemaLocation="../../common/provider.xsd" />
  <!-- this is mostly a chunk o' text. More normalized clinical notes will be in a diff schema -->
  <element name="SimpleClinicalNote">
    <complexType>
      <sequence>
        <element name="dateOfVisit" type="dateTime" minOccurs="1" maxOccurs="1" />
        <element name="finalizedAt" type="dateTime" minOccurs="0" maxOccurs="1" />
        <element name="visitType" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
        <element name="visitLocation" type="string" minOccurs="0" maxOccurs="1" />
        <element name="specialty" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
        <element name="signature" type="indivo:Signature" minOccurs="1" maxOccurs="unbounded" />
        <element name="chiefComplaint" type="string" minOccurs="0" maxOccurs="1" maxOccursUnbounded="1" />
        <element name="content" minOccurs="0" maxOccurs="1" type="string" />
      </sequence>
    </complexType>
  </element>
</schema>

Example:
<SimpleClinicalNote xmlns="http://indivo.org/vocab/xml/documents#">
  <dateOfVisit>2010-02-02T12:00:00Z</dateOfVisit>
  <finalizedAt>2010-02-03T13:12:00Z</finalizedAt>
  <visitType type="http://codes.indivo.org/visit-types#" value="acute">Acute Care</visitType>
  <visitLocation>Longfellow Medical</visitLocation>
  <specialty type="http://codes.indivo.org/specialties#" value="hem-onc">Hematology/Oncology</specialty>

  <signature>
    <at>2010-02-03T13:12:00Z</at>
    <provider>
      <name>Kenneth Mandl</name>
      <institution>Children's Hospital Boston</institution>
    </provider>
  </signature>

  <signature>
    <provider>
      <name>Isaac Kohane</name>
      <institution>Children's Hospital Boston</institution>
    </provider>
  </signature>

  <chiefComplaint>stomach ache</chiefComplaint>
  <content>
    Patient presents with ...
  </content>
</SimpleClinicalNote>

**Indivo Document Schema: SDMX**

For any *data model* in Indivo that can be represented in *SDML*, we will accept data in the form of *SDMX*.

Schema:

```xml
<complexType name="ModelType">
  <sequence>
    <element name="Field" minOccurs="0" maxOccurs="unbounded">
      <complexType mixed="true">
        <choice>
          <element name="Model" type="indivo:ModelType" minOccurs="0" maxOccurs="1"/>
          <element name="Models" type="indivo:ModelsType" minOccurs="0" maxOccurs="1"/>
        </choice>
        <attribute name="name" type="string" use="required"/>
      </complexType>
    </element>
  </sequence>
  <attribute name="name" type="string" use="required" />
  <attribute name="documentId" type="string" use="optional"/>
</complexType>

<complexType name="ModelsType">
  <sequence>
    <element name="Model" type="indivo:ModelType" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```
Adding Custom Schemas to Indivo

As of version 1.1 of Indivo X, we’ve added a feature that makes it much easier to add (in a drag-and-drop fashion) new supported schemas to an instance of Indivo. Adding a new schema to Indivo involves:

- Creating the schema
- Mapping the schema to Indivo’s Data Models
- Dropping the schema into the filesystem

Creating the Schema

Indivo currently accepts schemas only in XSD form. There are numerous tutorials and tools on the web to help you create an XSD, so we won’t presume to tell you how you should do it. What matters is that you build an XSD which can validate documents for further processing in the Indivo data pipeline.
Mapping the Schema to Data Models

In order to arrived at processed facts that can be queried and retrieved, you’ll need to a way to transform documents matching your schema into a form understood by Indivo. We call this tool (intuitively) a Transform, and you can learn how to build one [here](#).

If the data in your new schema doesn’t fit into any of the Indivo Data Models, and you want to add a new data-model to Indivo, see Adding Custom Data-Models to Indivo.

Dropping the Schema into the Filesystem

Indivo schemas currently have the following layout on the filesystem:

```
indivo_server/
    indivo/
        ...
    schemas/
        utils/
        metadata/
        data/
            common/
            output/
            core/
                sdmx/
                    schema.xsd
                    transform.[xslt | py]
                    sdmx.xml
                ...
        contrib/
```

The `indivo/schemas/data/core/` directory contains all of our built-in schemas, and you shouldn’t modify it. Since you are ‘contributing’ a schema to Indivo, add your schema to the `indivo/schemas/data/contrib/` directory. Simply:

- Create a new subdirectory under `indivo/schemas/data/contrib/`.
- Drop the following files into that directory:
  - `schema.xsd`: Your schema. This file **MUST BE NAMED SCHEMA.XSD** to be identified as a schema.
  - `transform.[xslt | py]`: Your transform. This file **MUST BE NAMED ‘transform’** to be identified.
  - `sample.xml` *(optional)*: A sample document that should validate against your schema. This is optional, but is a good way to make sure your schema works as intended. If you have one or more sample xml files in your directory (you can name them anything, as long as the filename ends in `.xml`), you can make sure that they all validate by running:

    ```
    cd indivo_server/indivo/schemas
    python utils/validate.py
    ```

    This will validate all sample documents against their schema: you should see ‘ok’ at the end of each line of output if there were no errors.
- Restart Indivo for your changes to take effect. Your final directory structure should now look something like:

  ```
  indivo_server/
      indivo/
          ...
      schemas/
  ```

3.17. Indivo Schemas
utils/
metadata/
data/
  common/
  output/
  core/
  sdmx/
    schema.xsd
    transform.[xslt | py]
    sdmx.xml
  ...
contrib/
  your_schema/
    schema.xsd
    transform.[xslt | py]
    your_example1.xml
    your_example2.xml
  ...
You’ll probably want to start with our *Installation Instructions*.

## 4.1 Architecture Overview

This document provides a basic overview of the Indivo X system architecture. This document should be read before continuing on to *Indivo Authentication* and *the Indivo API*.

### 4.1.1 Basic Indivo Concepts

**Indivo Record**: the complete set of medical information stored by Indivo about a single individual.

**Indivo Account**: a username/password to log into Indivo. One account may be able to access any number of Indivo Records, and one Indivo Record may be accessible by multiple Indivo Accounts.

**Indivo Document**: a piece of medical information stored in an Indivo Record.

### 4.1.2 Components

Indivo X comprises multiple components, each running as its own web server. Small installations may choose to install multiple components on a single physical server. The Indivo X Server is the core of the system; other components, including the Indivo User Interface, can be easily substituted by custom implementations.
Indivo X Server

For a given Indivo installation, the Indivo X server:
- stores all Indivo account information, as well as the medical records and documents,
- is responsible for authentication and authorization before granting access to Indivo data,
- exposes an API for access by administrative and user applications, and by the Indivo User Interface.

Indivo User Interface / Indivo Chrome

The Indivo User Interface, also known as the “Indivo Chrome”, implements the web-based visual interface that an Indivo user will view and use. The branding/colors/details of the user interface are all controlled by the Indivo Chrome. Indivo Chrome connects to Indivo X using the standard Indivo API, including some specific calls accessible only to the Chrome component.

Indivo X will ship with a default implementation of the Indivo Chrome which can be customized while maintaining a clean interface to the Indivo X API. Customizations are encouraged for re-branding or for entirely different devices, e.g. iPhone.

(The term “Chrome” is often used to describe the visual portions of a web browser that are part of the web browser itself, and not part of the web site content, e.g. the back button. Here, with “Indivo Chrome”, we mean the Indivo user interface that is part of the core Indivo service, not part of a user application that extends Indivo functionality.)
Administrative Application

An Admin Application can connect to Indivo X and

- create new Indivo accounts and records
- reset of passwords
- manage ownership of records, i.e. assigning an account as the owner of a record.

An admin application cannot access medical data, it can only manage a record’s metadata. An admin app is thus ideal for a hospital administrator, an Indivo help-desk staffer, a research administrator, etc.

User Application / Personal Health Application

A user application, or Personal Health Application, is an application that Indivo users manually add to their record to provide incremental functionality. Examples of PHA functionality include:

- Diabetes management
- Genomic data display
- Clinical trial matching and messaging

User applications generally provide a web interface to the Indivo user, while connecting to the user’s Indivo record directly with the Indivo X Server. Users are fully in control of what data a user application can access. They can, at any time, change those permissions or remove the application entirely. Thus, an Indivo user application connects to Indivo in much the same way that a Facebook application connects to Facebook.

4.1.3 Communication Protocols

All communication between components is over HTTPS, with an API that abides by the REST design philosophy. Authentication is via oAuth.

More about Indivo Authentication.

More about the Indivo API.

4.2 The Audit System

Auditing will live in a middleware that is called after the Authentication and Authorization middleware.

The ordering is so that auditing will know the principal and view_func/view_(kw)args during processing.

When no principal is present we will not audit.

If principal is not None then we will record relevant information related to the request.

The amount of information audited and which calls to audit will be configurable in the top-level settings.py file

4.2.1 Data to Audit

Redundancy is built into the audit table so that as few assumptions are made when investigating a req/resp. as possible. The proposed table will be broken down into five sections:
Basic Info

This information will always be audited, and contains basic information about the request.

- **datetime**: When the request was made
- **view_func**: Which Indivo View Function was called by the request
- **request_successful**: Was the request successful, or were there errors in its execution

Principal Info

Information about the principal of the request.

- **effective_principal_email**: The email of the principal making the request
- **proxied_by_email**: The email of the principal proxied by the principal making the request (i.e., the email of the Account being proxied by a PHA)

Resources

Information about the resources being accessed by a request

- **carenet_id**: Identifies the carenet a request is made through
- **record_id**: Identifies the record a request is made through
- **pha_id**: Identifies the PHA a request modifies
- **document_id**: Identifies the document accessed by a request
- **external_id**: Identifies the external identifier used to represent a resource in a request
- **message_id**: Identifies the message accessed by a request

Request Info

Information carried in by the request

- **req_url**: The URL to which the request was made
- **req_ip_address**: The ip adress from which the request originated
- **req_domain**: The domain from which the request originated
- **req_headers**: Headers associated with the request, including Oauth credentials
- **req_method**: The HTTP method with which the request was sent (GET, PUT, POST, DELETE)

Response Info

Information about the response being returned

- **resp_code**: The HTTP response code returned by the request
- **resp headers**: Headers sent out with the response, including content type
4.2.2 Configuring the Audit System

The following fields in settings.py may be altered to configure the system:

- **AUDIT_LEVEL**: Values are HIGH, MED, LOW, NONE. Controls the amount of information recorded for each call. See Audit Levels below for details.
- **AUDIT_OAUTH**: Values are True, False. Controls whether or not calls related to the oauth dance are audited.
- **AUDIT_FAILURE**: Values are True, False. Controls whether or not calls which exited with failure (HTTP Response Code of 400 or greater) are audited.

Default Values

Defaults for the audit settings record as much information as possible. They are:

- **AUDIT_LEVEL**: HIGH
- **AUDIT_OAUTH**: True
- **AUDIT_FAILURE**: True

Audit Levels

The information recorded at each audit level is also configurable. Defaults are:

- **NONE**: no information is recorded.
- **LOW**: Basic Information and Principal Information are recorded.
- **MED**: Basic Information, Principal Information, and Resources are recorded.
- **HIGH**: Basic Information, Principal Information, Resources, Request Information, and Response information are recorded.

4.2.3 Querying the Audit System

There will be a new API call that implements the API Query Interface. Calls to:

GET /records/{record_id}/audits/query/

May retrieve audit entries filtered and grouped by the following fields:

- **document_id**: The document modified by the request. **String**
- **external_id**: The external id used to reference a resource in the request. **String**
- **request_date**: The date on which the request was made. **Date**
- **function_name**: The internal Indivo X view function called by the request. **String**
- **principal_email**: The email of the principal making the request. **String**
- **proxied_by_email**: The email of the principal proxied by the principal making the request (i.e., the email of the Account being proxied by a PHA). **String**

The default ordering on results will be in descending order by request_date.

Audit queries will be returned (like all data returned form the API Query Interface) according to the *Indivo Reporting Schema*. Individual items will validate against the *Indivo Audit Log Schema*. 

4.2. The Audit System
4.2.4 Compatibility Issues with old Audit System

The Beta1 interface calls will be preserved, although they are now deprecated. However, in order to accommodate the new data being stored, the output schema for audit logs has changed. Audit logs will now be returned according to the Indivo Reporting Schema and new Indivo Audit Log Schema, as with the new call above.

4.3 Coding Systems

For Indivo, and in general for a number of health applications, coding systems are used for interoperability. Examples include vaccine disease codes, allergy codes, procedure codes, etc. This page documents a web-based mechanism for documenting and publishing, in a machine-readable manner, these coding systems.

4.3.1 Abstract Model

Beyond the basic attributes (name, publisher, description), a coding system includes:

- a way to list all codes
- a way to look up a single code
- a way to search for codes matching a simple text query (e.g. “diab” should match “diabetes.”)

A single code entry will have, at least:

- a code
- an abbreviation
- a full title
- (optionally) a description
- (optionally) relationships to other codes.

4.3.2 Data Representation

JSON

4.3.3 RESTful Calls

The URL templates define RESTful calls to obtain a single code, and to search for a number of codes. Specifically, given the example above, the following URL returns a single code “123”:

http://codes.indivo.org/systems/allergies/123

And the following URL searches the list of allergy codes for “peanut”:

http://codes.indivo.org/systems/allergies/search?q=peanut

4.3.4 Sources

The coding systems used in Indivo X are as follows. Individual installations need to download the coding systems on their own, as the licenses for these do not permit redistribution, so we cannot package them with Indivo.
Immunizations: HL7 v3

The easiest way to get the HL7 V3 file in vertical-bar-separated format, as required by the codingystem loader, is to use bioontology.org.

We specifically used the REST service at http://rest.bioontology.org/bioportal. The ontology code we used to download our version appears to no longer exist, so we’ll look into the latest codes soon. In the meantime, documentation for the REST service is at http://www.bioontology.org/wiki/index.php/NCBO_REST_services

Labs: LOINC

Available at http://loinc.org

There is an encoding issue which forced us to truncate the LOINC file for now at line 43504.

Problems: SNOMED CT


An encoding conversion is required to get to utf8, should be doable using the iconv program on most Linux installations.

Medications: RxTerms


Note that we may move to RxNorm instead of RxTerms.

4.4 Connecting Indivo to a Hospital

To connect Indivo to a hospital data feed, you need:

- A hospital that provides an appropriate interface we like to call “Point-of-Care API” (we’ve previously called this Hospital API, but it’s not just hospitals.)
- An Indivo app that reads that data feed and pushes the data into Indivo.

To help this along, we’ve produced:

- the Indivo app in question, which we call Indivo Hospital Connector, and
- a “Mock Hospital”, effectively a fake hospital that implements the Point-of-Care API we think most hospitals should provide.

Once a point of care exposes the Point of Care API, it should then be possible for any PCHR (Indivo, Google Health, Microsoft HealthVault) to connect to the Hospital and lead a user through the data-download process. Of course, this process requires “patient identity proofing”, where the point-of-care will ensure that the person connecting is accessing their data alone.
4.4.1 PoC API

The PoC API is REST API with oAuth used to generate an access token that can then be used for long-term access to the Point-of-care record that needs to be integrated with a PCHR. In every oAuth transaction, the user must authenticate to the data source and authorize access to the connector application (all components are diagrammed above.) Each Point of Care can authenticate users in the way it sees fit: with an account, with one-time credentials, with personal questions only the right patient can answer, etc... this decision is orthogonal to the oAuth process and does not need to be specified for a connector app to function properly. The authentication process, whatever it is, should be implemented as part of the Point of Care’s oAuth log-in-and-authorize process. In addition, each Point of Care can choose which connector apps it wants to enable, by providing only those approved connector apps with oAuth consumer tokens. This means that a Point of Care can implement the PoC API, but only allow specific PCHR}s to connect, once proper privacy/HIPAA arrangements have been made with each. In other words, the protocols are independent of the policy decisions regarding which apps can connect and how users are authenticated: points of care retain full control at all
At the end of the oAuth process, a PoC is expected to return the normal oAuth parameters: `oauth_token` and `oauth_token_secret`, and an additional parameter `xoauth_hospital_record_id`, which identifies the hospital record that has been connected to the token in question. This record ID is needed to make the appropriate REST call to obtain data.

In the PoC API, data is provided in individual documents. The API is then really simple: an Atom feed of documents, and a REST call for each document. Each Point of Care can choose which data format it supports. We encourage Points of Care to use the Indivo Schemas.

```
GET /data/{hospital_record_id}/documents/  
    { atom feed of documents }

GET /data/{hospital_record_id}/documents/{document_id}  
    { a single document of data, using identifiers provided in the atom feed }
```

### 4.4.2 Mock Hospital

The Mock Hospital software we provide implements the PoC REST API. Patient identity proofing is done using the patient’s last name and a confirmation code. In a real-world setting, this confirmation code could be provided in person to the patient, or by mail. It might be preferable to ask the patient to enter more than just their last name, too, and to lock down access if too many incorrect confirmation codes are entered. We do not concern ourselves with this for now: we simply show feasibility of patient-proofing during the oAuth process.

- Download MockHospital v0.1
- Untar in `/web/mock_hospital`
- Like Indivo, create a PostgreSQL database, preferably called mockhospital, with appropriate PG username access.
- Load the data:
  
  ```
  ./reset.sh
  ```

  This will load up data included in the `data/` directory, which you can edit to your preference before setting up mockhospital.

  Check out `/web/mock_hospital/hospital/dataimport.py` to understand the XML and file structure format of the `data/` directory

- Run the service:
  
  ```
  python manage.py runserver 7000
  ```

### 4.4.3 Indivo Hospital Connector

Indivo Hospital Connector is an Indivo app that also functions as an oAuth client to Mock Hospital (or to any PoC-API-compliant endpoint).

- Download Indivo Hospital Connector
- Set up in `/web/indivo_hospital_connector`
- Check `settings.py` for the correct connectivity parameters to Mock Hospital and Indivo.
- Run `./reset.sh` to create and initialize the “connector” database and database user
- Run the app:
Periodically (cron) run the following command to load docs from Mock Hospital to Indivo:

```
python manage.py load_documents
```

## 4.5 Indivo Data Models

### 4.5.1 Introduction

Data Models in Indivo describe the format in which Indivo represents medical information. They are **NOT** the same as Schemas, which describe formats that Indivo recognizes as valid input data. Rather, data models describe the final processed state of medical data in Indivo: how data are stored, how they are queryable via the Query API, and how they are returned via the Reporting API.

We also introduce one additional term: Medical Facts. A Fact is one datapoint corresponding to a data model: for example, a latex allergy is a Fact that is an instance of the Allergy data model. Internally, Indivo represents facts as Python objects, so you’ll see us referencing medical facts as fact objects as well.

### 4.5.2 Defining a Data Model

At its most basic level, a data model definition is just a list of fields and their types. For example, our Problem data model is defined as (some fields omitted):

- `date_onset`: Date
- `date_resolution`: Date
- `name`: String
- `comments`: String
- `diagnosed_by`: String

This is pretty simple, and we’d like to enable others add new data models to Indivo just as easily. So we currently allow two formats for defining data models:

#### Django Model Classes

Since our data models are directly mapped to database tables using Django’s ORM, they are most effectively represented as Django Models. Django has a flexible, powerful method for expressing fields as python class attributes, so data models defined in this way can harness the full capabilities of the Django ORM. Of course, representing data models in this way requires some knowledge of python. For a full reference of Django models, see Django models and Django model fields.

One important Indivo-specific note: when defining Django Model Classes, make sure to subclass `indivo.models.Fact`, which will ensure that your class can be treated as a data model. For example, your class definition might look like:

```
from indivo.models import Fact
from django.db import models

class YourModel(Fact):
    your_field1 = models.CharField(max_length=200, null=True)
```
Custom Django Model Fields

For modeling medical data, Indivo provides some custom Field Subclasses. These fields represent their data as multiple separate database fields, with names formed from the original field’s name and some appended suffixes (see the classes below for some examples). You should use these fields as if they were any other Django Model Field:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import YourFavoriteFieldSubclass

class YourModel(Fact):
    normal_field = models.CharField(max_length=200, null=True)
    special_field = YourFavoriteFieldSubclass()
```

Now YourModel has both a standard CharField, and also other fields defined by the Field Subclass. We define the following Field Subclasses:

```python
class indivo.fields.CodedValueField(Type)
    A field for representing coded data elements.
    Creating a CodedValueField named ‘value’, for example, will (under the hood) create thee fields:
    • value_identifier, the system-specific identifier that represents the element (i.e. an RXNorm CUI)
    • value_title, the human-readable title of the element
    • value_system, the coding system used to represent the element

    When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original value field name.

class indivo.fields.ValueAndUnitField(Type)
    A field for representing data elements with both a value and a unit.
    Creating a ValueAndUnitField named ‘frequency’, for example, will (under the hood) create the fields:
    • frequency_value, the value of the element
    • frequency_unit, the units in which the value is measured

    When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original frequency field name.

class indivo.fields.AddressField(Type)
    A field for representing a physical address.
    Creating an AddressField named ‘address’, for example, will (under the hood) create the fields:
    • address_country, the country in which the address is located
    • address_city, the city in which the address is located
    • address_postalcode, the postalcode of the address
    • address_region, the region (state, in the US) in which the address is located
    • address_street, the street address (including street number, apartment number, etc.) at which the address is located
```

4.5. Indivo Data Models
When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original address field name.

class indivo.fields.NameField

A field for representing a person’s name.

Creating a NameField named ‘name’, for example, will (under the hood) create the fields:

• name_family, the family (last) name of the person
• name_given, the given (first) name of the person
• name_middle, the middle name of the person
• name_prefix, the prefix (i.e. ‘Mr.’, ‘Sir’, etc.) for the person’s name
• name_suffix, the suffix (i.e. ‘Jr.’, ‘Ph.D.’, etc.) for the person’s name

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original name field name.

class indivo.fields.TelephoneField

A field for representing a telephone number.

Creating a TelephoneField named ‘phone’, for example, will (under the hood) create the fields:

• phone_type, The type of the phone number, limited to h (home), w (work), or c (cell)
• phone_number, The actual phone number
• phone_preferred_p, Whether or not this number is a preferred method of contact (True or False)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original phone field name.

class indivo.fields.PharmacyField

A field for representing a pharmacy.

Creating a PharmacyField named ‘pharmacy’, for example, will (under the hood) create three fields:

• pharmacy_ncpdpid, the pharmacy’s National Council for Prescription Drug Programs (NCPDP) ID number
• pharmacy_adr, the address at which the pharmacy is located (an AddressField)
• pharmacy_org, the name of the organization that owns the pharmacy

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original pharmacy field name.

class indivo.fields.ProviderField

A field for representing a medical provider.

Creating a ProviderField named ‘doc’, for example, will (under the hood) create the fields:

• doc_dea_number, the provider’s Drug Enforcement Agency (DEA) number
• doc_ethnicity, the provider’s ethnicity
• doc_npi_number, the provider’s National Provider Identification (NPI) number
• doc_preferred_language, the provider’s preferred language
• doc_race, the provider’s race
• doc_adr, the provider’s address (an AddressField)
• doc_bday, the provider’s birth date
• `doc_email`, the provider’s email address
• `doc_name`, the provider’s name (a NameField)
• `doc_tel_1`, the provider’s primary phone number (a TelephoneField)
• `doc_tel_2`, the provider’s secondary phone number (a TelephoneField)
• `doc_gender`, the provider’s gender, limited to `m` (male) or `f` (female)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `doc` field name.

class indivo.fields.VitalSignField (Type)
A field for representing a single measurement of a vital sign.

Creating a VitalSignField named ‘bp’, for example, will (under the hood) create the fields:

• `bp_unit`, the unit of the measurement
• `bp_value`, the value of the measurement
• `bp_name`, the name of the measurement (a CodedValueField)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `bp` field name.

class indivo.fields.BloodPressureField (Type)
A field for representing a blood pressure measurement.

Creating a BloodPressureField named ‘bp’, for example, will (under the hood) create the fields:

• `bp_position`, the position in which the measurement was taken (a CodedValueField)
• `bp_site`, the site on the body at which the measurement was taken (a CodedValueField)
• `bp_method`, the method of the measurement (a CodedValueField)
• `bp_diastolic`, the diastolic blood pressure (a VitalSignField)
• `bp_systolic`, the systolic blood pressure (a VitalSignField)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `bp` field name.

class indivo.fields.ValueRangeField (Type)
A field for representing a range of values.

Creating a ValueRangeField named ‘normal_range’, for example, will (under the hood) create the fields:

• `normal_range_max`, the maximum value of the range (a ValueAndUnitField)
• `normal_range_min`, the minimum value of the range (a ValueAndUnitField)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `normal_range` field name.

class indivo.fields.QuantitativeResultField (Type)
A field for representing a quantitative result, and expected ranges for that result.

Creating a QuantitativeResultField named ‘lab_result’, for example, will (under the hood) create the fields:

• `lab_result_non_critical_range`, the range outside of which results are ‘critical’ (a ValueRangeField)
• `lab_result_normal_range`, the range outside of which results are ‘abnormal’ (a ValueRangeField)
• lab_result_value, the actual result (a ValueAndUnitField)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original lab_result field name.

**Simple Data Modeling Language (SDML)**

For those less python-savvy who are still capable of thinking in terms of ‘fields’ and ‘types’ (which should be most people), we’ve defined a JSON-based modeling language for defining the very simple data models easily. SDML is less flexible than Django’s modeling language, but is much quicker to get started with and is less verbose for describing simple models. See our documentation of the language [here](#).

**Feeling Lost?**

For help getting started, see our core data models, below, each of which provide definitions both in SDML and Django Model classes.

### 4.5.3 Data Models and the Query API

Since the Query API allows app developers to directly apply filters and ranges to the datamodels they are selecting, they need to know what fields they are allowed to query against. The answer is simple:

**ANY FIELD ON A DATA MODEL THAT IS NOT A RELATION TO ANOTHER MODEL MAY BE USED IN THE QUERY API!**

For example, we introduced the ‘Problem’ model above, which has the fields:

- date_onset: Date
- date_resolution: Date
- name: String
- comments: String
- diagnosed_by: String

If you were making an API call such as GET /records/{RECORD_ID}/reports/minimal/problems/, you could filter by any of:

- date_onset
- date_resolution
- name
- comments
- diagnosed_by

If the problems model were a bit more complicated, and had another field:

- prescribed_med: Medication

You wouldn’t be able to filter by prescribed_med, since that field is a relation to another model.

The only exceptions to this rule are custom Django Model Fields. Such fields are translated into fields with other names, as described above. Any of these fields may be used in the query API, but (for example), when looking at a model with a CodedValue element such as:

- problem_type: CodedValue
You would be able to filter by `problem_type_identifier`, `problem_type_title`, or `problem_type_system`, but not by `problem_type` itself.

### 4.5.4 Core Data Models

Here is a listing of the data models currently supported by Indivo. Each instance might define other, contributed models: see below for information on how to add data models to Indivo.

**Indivo Data Model: Allergy**

**Model Definition**

As SDML:

```
[{
   "__modelname__": "Allergy",
   "allergic_reaction": "CodedValue",
   "category": "CodedValue",
   "drug_allergen": "CodedValue",
   "drug_class_allergen": "CodedValue",
   "food_allergen": "CodedValue",
   "severity": "CodedValue"
},
{
   "__modelname__": "AllergyExclusion",
   "name": "CodedValue"
}]
```

As a Django Model Class:

```python
from indivo.models import Fact
from indivo.fields import CodedValueField

class Allergy(Fact):
    allergic_reaction = CodedValueField()
    category = CodedValueField()
    drug_allergen = CodedValueField()
    drug_class_allergen = CodedValueField()
    food_allergen = CodedValueField()
    severity = CodedValueField()

class AllergyExclusion(Fact):
    name = CodedValueField()
```

**Examples**

As SDMJ:

```
[{
   "__modelname__": "Allergy",
   "allergic_reaction_title": "Anaphylaxis",
   "allergic_reaction_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
   "allergic_reaction_identifier": "39579001",
   "category_title": "Drug allergy",
   "category_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
   "category_identifier": "5618005",
   "allergic_reaction_identifier": "4277005",
}
```
"category_identifier": "416098002",
"drug_class_allergen_title": "Sulfonamide Antibacterial",
"drug_class_allergen_system": "http://purl.bioontology.org/ontology/NDFRT/",
"drug_class_allergen_identifier": "N0000175503",
"severity_title": "Severe",
"severity_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
"severity_identifier": "24484000"
},
{
  "__modelname__": "AllergyExclusion",
  "name_title": "No known allergies",
  "name_identifier": "160244002",
  "name_system": "http://purl.bioontology.org/ontology/SNOMEDCT"
}
]

As SDMX:

```xml
<Models>
  <Model name="Allergy">
    <Field name="allergic_reaction_title">Anaphylaxis</Field>
    <Field name="allergic_reaction_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="allergic_reaction_identifier">39579001</Field>
    <Field name="category_title">Drug allergy</Field>
    <Field name="category_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="category_identifier">416098002</Field>
    <Field name="drug_class_allergen_title">Sulfonamide Antibacterial</Field>
    <Field name="drug_class_allergen_system">http://purl.bioontology.org/ontology/NDFRT/</Field>
    <Field name="drug_class_allergen_identifier">N0000175503</Field>
    <Field name="severity_title">Severe</Field>
    <Field name="severity_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="severity_identifier">24484000</Field>
  </Model>
  <Model name="AllergyExclusion">
    <Field name="name_title">No known allergies</Field>
    <Field name="name_identifier">160244002</Field>
    <Field name="name_system">http://purl.bioontology.org/ontology/SNOMEDCT</Field>
  </Model>
</Models>

As a Fact object:

```python
from indivo.models import Allergy

allergy_fact = Allergy(
  allergic_reaction_title="Anaphylaxis",
  allergic_reaction_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
  allergic_reaction_identifier="39579001",
  category_title="Drug allergy",
  category_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
  category_identifier="416098002",
  drug_class_allergen_title="Sulfonamide Antibacterial",
  drug_class_allergen_system="http://purl.bioontology.org/ontology/NDFRT/",
  drug_class_allergen_identifier="N0000175503",
  severity_title="Severe",
  severity_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
  severity_identifier="24484000",
)

allergy_exclusion = AllergyExclusion(
```
Indivo Data Model: Equipment

Model Definition

As SDML:

```json
{
    "__modelname__": "Equipment",
    "date_started": "Date",
    "date_stopped": "Date",
    "name": "String",
    "vendor": "String",
    "description": "String"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class Equipment(Fact):
    date_started = models.DateField(null=True)
    date_stopped = models.DateField(null=True)
    name = models.CharField(max_length=40)
    vendor = models.CharField(max_length=40, null=True)
    description = models.TextField(null=True)
```

Examples

As SDMJ:

```json
{
    "__modelname__": "Equipment",
    "date_started": "2009-02-05",
    "date_stopped": "2009-06-12",
    "name": "Pacemaker",
    "vendor": "Acme Medical Devices",
    "description": "it works!"
}
```

As SDMX:

```xml
<Models>
    <Model name="Equipment">
        <Field name="dateStarted">2009-02-05</Field>
        <Field name="dateStopped">2009-06-12</Field>
        <Field name="name">Pacemaker</Field>
        <Field name="vendor">Acme Medical Devices</Field>
        <Field name="description">it works!</Field>
    </Model>
</Models>
```
As a Fact object:

```python
from indivo.models import Equipment
from indivo.lib.iso8601 import parse_utc_date as date

equipment_fact = Equipment(
    date_started=date("2009-02-05"),
    date_stopped=date("2009-06-12"),
    name="Pacemaker",
    vendor="Acme Medical Devices",
    description="it works!"
)
```

Indivo Data Model: Immunization

Model Definition

As SDML:

```json
{
    "__modelname__": "Immunization",
    "date": "Date",
    "administration_status": "CodedValue",
    "product_class": "CodedValue",
    "product_class_2": "CodedValue",
    "product_name": "CodedValue",
    "refusal_reason": "CodedValue"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField

class Immunization(Fact):
    date = models.DateTimeField(null=True)
    administration_status = CodedValueField()
    product_class = CodedValueField()
    product_class_2 = CodedValueField()
    product_name = CodedValueField()
    refusal_reason = CodedValueField()
```

Examples

As SDMJ:

```json
{
    "__modelname__": "Immunization",
    "date": "2009-05-16T12:00:00Z",
    "administration_status_title": "Not Administered",
    "administration_status_system": "http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#",
    "administration_status_identifier": "notAdministered",
    "product_class_title": "TYPHOID",
    "product_class_system": "http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#",
    "product_class_identifier": "TYPHOID",
}
```
As SDMX:

```xml
<Models>
  <Model name="Immunization">
    <Field name="date">2009-05-16T12:00:00Z</Field>
    <Field name="administration_status_title">Not Administered</Field>
    <Field name="administration_status_system">http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#</Field>
    <Field name="administration_status_identifier">notAdministered</Field>
    <Field name="product_class_title">TYPHOID</Field>
    <Field name="product_class_system">http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#</Field>
    <Field name="product_class_identifier">TYPHOID</Field>
    <Field name="product_name_title">typhoid, oral</Field>
    <Field name="product_name_system">http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx#</Field>
    <Field name="product_name_identifier">25</Field>
    <Field name="refusal_reason_title">Allergy to vaccine/vaccine components, or allergy to eggs</Field>
    <Field name="refusal_reason_system">http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#</Field>
    <Field name="refusal_reason_identifier">allergy</Field>
  </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import Immunization
from indivo.lib.iso8601 import parse_utc_date as date

immunization_fact = Immunization(
    date=date("2009-05-16T12:00:00Z"),
    administration_status_title="Not Administered",
    administration_status_system="http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#",
    administration_status_identifier="notAdministered",
    product_class_title="TYPHOID",
    product_class_system="http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#",
    product_class_identifier="TYPHOID",
    product_name_title="typhoid, oral",
    product_name_system="http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx#",
    product_name_identifier="25",
    refusal_reason_title="Allergy to vaccine/vaccine components, or allergy to eggs",
    refusal_reason_system="http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#",
    refusal_reason_identifier="allergy",
)
```

**Indivo Data Model: LabResult**

**Model Definition**

**As SDML:**

```json
"__modelname__": "LabResult",
```
"abnormal_interpretation": "CodedValue",
"accession_number": "String",
"test_name": "CodedValue",
"status": "CodedValue",
"narrative_result": "String",
"notes": "String",
"quantitative_result": "QuantitativeResult",
"collected_at": "Date",
"collected_by_org": "Organization",
"collected_by_name": "Name",
"collected_by_role": "String"
}

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField, QuantitativeResultField, OrganizationField, NameField

class LabResult(Fact):
    abnormal_interpretation = CodedValueField()
    accession_number = models.CharField(max_length=255, null=True)
    test_name = CodedValueField()
    status = CodedValueField()
    narrative_result = models.CharField(max_length=255, null=True)
    notes = models.CharField(max_length=255, null=True)
    quantitative_result = QuantitativeResultField()
    collected_at = models.DateTimeField(null=True)
    collected_by_org = OrganizationField()
    collected_by_name = NameField()
    collected_by_role = models.CharField(max_length=255, null=True)
```

Examples

As SDMJ:

```json
{
    "__modelname__": "LabResult",
    "abnormal_interpretation_title": "Normal",
    "abnormal_interpretation_system": "http://smartplatforms.org/terms/codes/LabResultInterpretation#",
    "abnormal_interpretation_identifier": "normal",
    "accession_number": "AC09205823577",
    "test_name_title": "Serum Sodium",
    "test_name_system": "http://purl.bioontology.org/ontology/LNC/",
    "test_name_identifier": "2951-2",
    "status_title": "Final results": complete and verified",
    "status_system": "http://smartplatforms.org/terms/codes/LabStatus#",
    "status_identifier": "final",
    "notes": "Blood sample appears to have hemolyzed",
    "quantitative_result_non_critical_range_max_value": "155",
    "quantitative_result_non_critical_range_max_unit": "mEq/L",
    "quantitative_result_non_critical_range_min_value": "120",
}```
"quantitative_result_non_critical_range_min_unit": "mEq/L",
"quantitative_result_normal_range_max_value": "145",
"quantitative_result_normal_range_max_unit": "mEq/L",
"quantitative_result_normal_range_min_value": "135",
"quantitative_result_normal_range_min_unit": "mEq/L",
"quantitative_result_value_value": "140",
"quantitative_result_value_unit": "mEq/L",
"collected_at": "2010-12-27T17:00:00Z",
"collected_by_org_name": "City Lab",
"collected_by_org adr_country": "USA",
"collected_by_org adr_city": "Springfield",
"collected_by_org adr_postalcode": "11111",
"collected_by_org adr_region": "MA",
"collected_by_org adr street": "20 Elm St",
"collected_by name family": "Finnialispi",
"collected_by name given": "Tad",
"collected_by role": "Lab Specialist"
}

As SDMX:

<Models>
  <Model name="LabResult">
    <Field name="abnormal_interpretation_title">Normal</Field>
    <Field name="abnormal_interpretation_system">http://smartplatforms.org/terms/codes/LabResultInterpretation#</Field>
    <Field name="abnormal_interpretation_identifier">normal</Field>
    <Field name="accession_number">AC09205823577</Field>
    <Field name="test_name_title">Serum Sodium</Field>
    <Field name="test_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
    <Field name="test_name_identifier">2951-2</Field>
    <Field name="status_title">Final results: complete and verified</Field>
    <Field name="status_system">http://smartplatforms.org/terms/codes/LabStatus#</Field>
    <Field name="status_identifier">final</Field>
    <Field name="notes">Blood sample appears to have hemolyzed</Field>
    <Field name="quantitative_result_non_critical_range_max_value">155</Field>
    <Field name="quantitative_result_non_critical_range_max_unit">mEq/L</Field>
    <Field name="quantitative_result_non_critical_range_min_value">120</Field>
    <Field name="quantitative_result_non_critical_range_min_unit">mEq/L</Field>
    <Field name="quantitative_result_normal_range_max_value">145</Field>
    <Field name="quantitative_result_normal_range_max_unit">mEq/L</Field>
    <Field name="quantitative_result_normal_range_min_value">135</Field>
    <Field name="quantitative_result_normal_range_min_unit">mEq/L</Field>
    <Field name="quantitative_result_value_value">140</Field>
    <Field name="quantitative_result_value_unit">mEq/L</Field>
    <Field name="collected_at">2010-12-27T17:00:00Z</Field>
  </Model>
</Models>
As a Fact object:

```python
from indivo.models import LabResult
from indivo.lib.iso8601 import parse_utc_date as date

lab_fact = LabResult(
    abnormal_interpretation_title="Normal",
    abnormal_interpretation_system="http://smartplatforms.org/terms/codes/LabResultInterpretation#",
    abnormal_interpretation_identifier="normal",

    accession_number="AC09205823577",
    test_name_title="Serum Sodium",
    test_name_system="http://purl.bioontology.org/ontology/LNC/",
    test_name_identifier="2951-2",

    status_title="Final results: complete and verified",
    status_system="http://smartplatforms.org/terms/codes/LabStatus#",
    status_identifier="final",

    notes="Blood sample appears to have hemolyzed",

    quantitative_result_non_critical_range_max_value="155",
    quantitative_result_non_critical_range_max_unit="mEq/L",
    quantitative_result_non_critical_range_min_value="120",
    quantitative_result_non_critical_range_min_unit="mEq/L",

    quantitative_result_normal_range_max_value="145",
    quantitative_result_normal_range_max_unit="mEq/L",
    quantitative_result_normal_range_min_value="135",
    quantitative_result_normal_range_min_unit="mEq/L",

    quantitative_result_value_value="140",
    quantitative_result_value_unit="mEq/L",

    collected_at=date("2010-12-27T17:00:00Z"),
    collected_by_org_name="City Lab",
    collected_by_org_adr_country="USA",
    collected_by_org_adr_city="Springfield",
    collected_by_org_adr_postalcode="11111",
    collected_by_org_adr_region="MA",
    collected_by_org_adr_street="20 Elm St",
```
Indivo X Documentation, Release 2.0.0

Indivo Data Model: Medication

Model Definition

As SDML:

```json
{  
  "__modelname__": "Medication",
  "drugName": "CodedValue",
  "endDate": "Date",
  "frequency": "ValueAndUnit",
  "instructions": "String",
  "provenance": "CodedValue",
  "quantity": "ValueAndUnit",
  "startDate": "Date",
  "fulfillments": [{
    "__modelname__": "Fill",
    "date": "Date",
    "dispenseDaysSupply": "Number",
    "pbm": "String",
    "pharmacy": "Pharmacy",
    "provider": "Provider",
    "quantityDispensed": "ValueAndUnit"
  }]
}
```

**Note:** Since SDML doesn’t provide for Boolean Fields, we are unable to define the `dispense_as_written` field properly in SDML. Our actual implementation of the Medication data model uses a Django Model Class for this reason.

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField, ValueAndUnitField, PharmacyField, ProviderField

class Medication(Fact):
    drugName = CodedValueField()
    endDate = models.DateField(null=True)
    frequency = ValueAndUnitField()
    instructions = models.CharField(max_length=255, null=True)
    provenance = CodedValueField()
    quantity = ValueAndUnitField()
    startDate = models.DateField(null=True)

class Fill(Fact):
    date = models.DateTimeField(null=True)
    dispenseDaysSupply = models.FloatField(null=True)
    pbm = models.CharField(max_length=255, null=True)
    pharmacy = PharmacyField()
    provider = ProviderField()
```

4.5. Indivo Data Models 341
quantityDispensed = ValueAndUnitField()
medication = models.ForeignKey(Medication, null=True, related_name='fulfillments')

Examples

As SDMJ:

```json
{
  "__modelName__": "Medication",
  "drugName_title": "AMITRIPTYLINE HCL 50 MG TAB",
  "drugName_system": "http://purl.bioontology.org/ontology/RXNORM/",
  "drugName_identifier": "856845",
  "endDate": "2007-08-14",
  "frequency_value": "2",
  "frequency_unit": "/d",
  "instructions": "Take two tablets twice daily as needed for pain",
  "provenance_title": "Derived by prescription",
  "provenance_system": "http://smartplatforms.org/terms/codes/MedicationProvenance#",
  "provenance_identifier": "prescription",
  "quantity_value": "2",
  "quantity_unit": "{tablet}",
  "startDate": "2007-03-14",
  "fulfillments": [
    {
      "__modelName__": "Fill",
      "date": "2007-03-14T04:00:00Z",
      "dispenseDaysSupply": "30",
      "pbm": "T00000000001011",
      "pharmacy_ncpdpid": "5235235",
      "pharmacy_org": "CVS #588",
      "pharmacy_adr_country": "Australia",
      "pharmacy_adr_city": "WonderCity",
      "pharmacy_adr_postalcode": "5555",
      "pharmacy_adr_street": "111 Lake Drive",
      "provider_dea_number": "325555555",
      "provider_npi_number": "5235235",
      "provider_email": "joshua.mandel@fake.emailserver.com",
      "provider_name_given": "Josuha",
      "provider_name_family": "Mandel",
      "provider_tel_1_type": "w",
      "provider_tel_1_number": "1-235-947-3452",
      "provider_tel_1_preferred": true,
      "quantityDispensed_value": "60",
      "quantityDispensed_unit": "{tablet}"
    },
    {
      "__modelName__": "Fill",
      "date": "2007-04-14T04:00:00Z",
      "dispenseDaysSupply": "30",
      "pbm": "T00000000001011",
      "pharmacy_ncpdpid": "5235235",
      "pharmacy_org": "CVS #588",
      "pharmacy_adr_country": "Australia",
      "pharmacy_adr_city": "WonderCity",
      "pharmacy_adr_postalcode": "5555",
      "pharmacy_adr_street": "111 Lake Drive",
      "provider_dea_number": "325555555",
```

Chapter 4. Indivo Administrators
As SDMX:

<Models>
  <Model name="Medication">
    <Field name="drugName_title">AMITRIPTYLINE HCL 50 MG TAB</Field>
    <Field name="drugName_system">http://purl.bioontology.org/ontology/RXNORM/</Field>
    <Field name="drugName_identifier">856845</Field>
    <Field name="endDate">2007-08-14</Field>
    <Field name="frequency_value">2</Field>
    <Field name="frequency_unit">/d</Field>
    <Field name="instructions">Take two tablets twice daily as needed for pain</Field>
    <Field name="provenance_title">Derived by prescription</Field>
    <Field name="provenance_system">http://smartplatforms.org/terms/codes/MedicationProvenance#</Field>
    <Field name="provenance_identifier">prescription</Field>
    <Field name="quantity_value">2</Field>
    <Field name="quantity_unit">{tablet}</Field>
    <Field name="startDate">2007-03-14</Field>
    <Fulfills>
      <Model name="Fill">
        <Field name="date">2007-03-14T04:00:00Z</Field>
        <Field name="dispenseDaysSupply">30</Field>
        <Field name="pbm">T00000000001011</Field>
        <Field name="pharmacy_ncpdpid">5235235</Field>
        <Field name="pharmacy_org">CVS #588</Field>
        <Field name="pharmacy_adr_country">Australia</Field>
        <Field name="pharmacy_adr_city">WonderCity</Field>
        <Field name="pharmacy_adr_postalcode">5555</Field>
        <Field name="pharmacy_adr_street">111 Lake Drive</Field>
        <Field name="provider_dea_number">325555555</Field>
        <Field name="provider_npi_number">5235235</Field>
        <Field name="provider_email">joshua.mandel@fake.emailserver.com</Field>
        <Field name="provider_name_given">Josuha</Field>
        <Field name="provider_name_family">Mandel</Field>
        <Field name="provider_tel_1_type">w</Field>
        <Field name="provider_tel_1_number">1-235-947-3452</Field>
        <Field name="provider_tel_1_preferred_p">true</Field>
        <Field name="quantityDispensed_value">60</Field>
        <Field name="quantityDispensed_unit">{tablet}</Field>
      </Model>
      <Model name="Fill">
        <Field name="date">2007-04-14T04:00:00Z</Field>
        <Field name="dispenseDaysSupply">30</Field>
        <Field name="pbm">T00000000001011</Field>
        <Field name="pharmacy_ncpdpid">5235235</Field>
      </Model>
    </Fulfills>
  </Model>
</Models>
As a Fact object:

```python
from indivo.models import Medication, Fill
from indivo.lib.iso8601 import parse_utc_date as date

med = Medication(
    drugName_title="AMITRIPTYLINE HCL 50 MG TAB",
    drugName_system="http://purl.bioontology.org/ontology/RXNORM/",
    drugName_identifier="856845",
    endDate=date("2007-08-14"),
    frequency_value="2",
    frequency_unit="/d",
    instructions="Take two tablets twice daily as needed for pain",
    provenance_title="Derived by prescription",
    provenance_system="http://smartplatforms.org/terms/codes/MedicationProvenance#",
    provenance_identifier="prescription",
    quantity_value="2",
    quantity_unit="{tablet}"
)

fill1 = Fill(
    date=date("2007-03-14T04:00:00Z"),
    dispenseDaysSupply=30,
    phbm="T000000000001011",
    pharmacy_ncpdpid="5235235",
    pharmacy_org="CVS #588",
    pharmacy_adr_country="Australia",
    pharmacy_adr_city="WonderCity",
    pharmacy_adr_postalcode="5555",
    pharmacy_adr_street="111 Lake Drive",
    provider_dea_number="325555555",
    provider_npi_number="5235235",
    provider_email="joshua.mandel@fake.emailserver.com",
    provider_name_given="Josuha",
    provider_name_family="Mandel",
    provider_tel_1_type="w",
    provider_tel_1_number="1-235-947-3452",
)```
provider_tel_1_preferred_p=True,
quantityDispensed_value="60",
quantityDispensed_unit="\{tablet\}"
)

fill12 = Fill(
  date=date("2007-04-14T04:00:00Z"),
dispenseDaysSupply=30,
  pbnm="T00000000001011",
  pharmacy_ncpdpid="5235235",
  pharmacy_org="CVS #588",
  pharmacy_adr_country="Australia",
  pharmacy_adr_city="WonderCity",
  pharmacy_adr_postalcode="5555",
  pharmacy_adr_street="111 Lake Drive",
  provider_dea_number="325555555",
  provider_npi_number="5235235",
  provider_email="joshua.mandel@fake.emailserver.com",
  provider_name_given="Josuha",
  provider_name_family="Mandel",
  provider_tel_1_type="w",
  provider_tel_1_number="1-235-947-3452",
  provider_tel_1_preferred_p=True,
  quantityDispensed_value="60",
  quantityDispensed_unit="\{tablet\}"
)

# save the medication so we can relate other objects to it
med.save()
med.fulfillments = [fill1, fill2]
med.save()

Indivo Data Model: Problem

Model Definition

As SDML:

{  
  "__modelname__": "Problem",
  "startDate": "Date",
  "endDate": "Date",
  "name": "CodedValue",
  "notes": "String"
}

As a Django Model Class:

```python
from indivo.models import Fact
from indivo.fields import CodedValueField
from django.db import models

class Problem(Fact):
    startDate = models.DateTimeField(null=True)
    endDate = models.DateTimeField(null=True)
    name = CodedValueField()
    notes = models.TextField(null=True)
```

4.5. Indivo Data Models
Examples

As SDMJ:

```json
{
   "__modelname__": "Problem",
   "startDate": "2009-05-16T12:00:00Z",
   "endDate": "2009-05-16T16:00:00Z",
   "name_title": "Backache (finding)",
   "name_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
   "name_identifier": "161891005"
}
```

As SDMX:

```xml
<Models>
   <Model name="Problem">
      <Field name="startDate">2009-05-16T12:00:00Z</Field>
      <Field name="endDate">2009-05-16T16:00:00Z</Field>
      <Field name="name_title">Backache (Finding)</Field>
      <Field name="name_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
      <Field name="name_identifier">161891005</Field>
   </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import Problem
from indivo.lib.iso8601 import parse_utc_date as date

problem_fact = Problem(
    startDate=date("2009-05-16T12:00:00Z"),
    endDate=date("2009-05-16T16:00:00Z"),
    name_title="Backache (finding)",
    name_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    name_identifier="161891005",
)
```

Indivo Data Model: Procedure

Model Definition

As SDML:

```json
{
   "__modelname__": "Procedure",
   "date_performed": "Date",
   "name": "String",
   "name_type": "String",
   "name_value": "String",
   "name_abbrev": "String",
   "provider_name": "String",
   "provider_institution": "String",
   "location": "String",
   "comments": "String"
}
```
As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class Procedure(Fact):
    date_performed = models.DateTimeField(null=True)
    name = models.CharField(max_length=100)
    name_type = models.CharField(max_length=80, null=True)
    name_value = models.CharField(max_length=40, null=True)
    name_abbrev = models.CharField(max_length=20, null=True)
    provider_name = models.CharField(max_length=200, null=True)
    provider_institution = models.CharField(max_length=200, null=True)
    location = models.CharField(max_length=100, null=True)
    comments = models.TextField(null=True)
```

Examples

As SDMJ:

```json
{
    "__modelname__": "Procedure",
    "date_performed": "2009-05-16T12:00:00",
    "name": "Appendectomy",
    "name_type": "http://codes.indivo.org/procedures#",
    "name_value": "123",
    "name_abbrev": "append",
    "provider_name": "Kenneth Mandl",
    "provider_institution": "Children’s Hospital Boston",
    "location": "300 Longwood Ave, Boston MA 02115",
    "comments": "Went great!"
}
```

As SDMX:

```xml
<Models>
    <Model name="Procedure">
        <Field name="date_performed">2009-05-16T12:00:00</Field>
        <Field name="name">Appendectomy</Field>
        <Field name="name_type">http://codes.indivo.org/procedures#</Field>
        <Field name="name_value">123</Field>
        <Field name="name_abbrev">append</Field>
        <Field name="provider_name">Kenneth Mandl</Field>
        <Field name="provider_institution">Children’s Hospital Boston</Field>
        <Field name="location">300 Longwood Ave, Boston MA 02115</Field>
        <Field name="comments">Went great!</Field>
    </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import Procedure
from indivo.lib.iso8601 import parse_utc_date as date

procedure_fact = Procedure(
    date_performed=date("2009-05-16T12:00:00"),
    name="Appendectomy",
    name_type="http://codes.indivo.org/procedures#",
```
Indivo Data Model: VitalSigns

Model Definition

As SDML:

```python

As a Django Model Class:

```python

```
provider = ProviderField()
encounterType = CodedValueField()

Examples

As SDMJ:

{
  "__modelname__": "VitalSigns",
  "date": "2009-05-16T12:00:00Z",
  "encounter": {
    "__modelname__": "Encounter",
    "startDate": "2009-05-16T12:00:00Z",
    "endDate": "2009-05-16T16:00:00Z",
    "facility_name": "Wonder Hospital",
    "facility_adr_country": "Australia",
    "facility_adr_city": "WonderCity",
    "facility_adr_postalcode": "5555",
    "facility_adr_street": "111 Lake Drive",
    "provider_dea_number": "3255555555",
    "provider_npi_number": "5235235",
    "provider_email": "joshua.mandel@fake.emailserver.com",
    "provider_name_given": "Josuha",
    "provider_name_family": "Mandel",
    "provider_tel_1_type": "w",
    "provider_tel_1_number": "1-235-947-3452",
    "provider_tel_1_preferred_p": true,
    "encounterType_title": "Ambulatory encounter",
    "encounterType_system": "http://smartplatforms.org/terms/codes/EncounterType#",
    "encounterType_identifier": "ambulatory"
  },
  "bp_position_title": "Sitting",
  "bp_position_identifier": "33586001",
  "bp_position_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
  "bp_site_title": "Right arm",
  "bp_site_identifier": "368209003",
  "bp_site_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
  "bp_method_title": "Auscultation",
  "bp_method_identifier": "auscultation",
  "bp_method_system": "http://smartplatforms.org/terms/codes/BloodPressureMethod#",
  "bp_diastolic_unit": "mm[Hg]",
  "bp_diastolic_value": 82,
  "bp_diastolic_name_title": "Intravascular diastolic",
  "bp_diastolic_name_identifier": "8462-4",
  "bp_diastolic_name_system": "http://purl.bioontology.org/ontology/LNC/",
  "bp_systolic_unit": "mm[Hg]",
  "bp_systolic_value": 132,
  "bp_systolic_name_title": "Intravascular systolic",
  "bp_systolic_name_identifier": "8480-6",
  "bp_systolic_name_system": "http://purl.bioontology.org/ontology/LNC/",
  "bmi_unit": "kg/m2",
  "bmi_value": 21.8,
  "bmi_name_title": "Body mass index",
  "bmi_name_system": "http://purl.bioontology.org/ontology/LNC/",
  "bmi_name_identifier": "39156-5",
}
"heart_rate_unit": "(beats)/min",
"heart_rate_value": 70,
"heart_rate_name_title": "Heart rate",
"heart_rate_name_system": "http://purl.bioontology.org/ontology/LNC/",
"heart_rate_name_identifier": "8867-4",

"height_unit": "m",
"height_value": 1.8,
"height_name_title": "Body height",
"height_name_system": "http://purl.bioontology.org/ontology/LNC/",
"height_name_identifier": "8302-2",

"oxygen_saturation_unit": "%{HemoglobinSaturation}'",
"oxygen_saturation_value": 99,
"oxygen_saturation_name_title": "Oxygen saturation",
"oxygen_saturation_name_system": "http://purl.bioontology.org/ontology/LNC/",
"oxygen_saturation_name_identifier": "2710-2",

"respiratory_rate_unit": "(breaths)/min",
"respiratory_rate_value": 16,
"respiratory_rate_name_title": "Respiration rate",
"respiratory_rate_name_system": "http://purl.bioontology.org/ontology/LNC/",
"respiratory_rate_name_identifier": "9279-1",

"temperature_unit": "Cel",
"temperature_value": 37,
"temperature_name_title": "Body temperature",
"temperature_name_system": "http://purl.bioontology.org/ontology/LNC/",
"temperature_name_identifier": "8310-5",

"weight_unit": "kg",
"weight_value": 70.8,
"weight_name_title": "Body weight",
"weight_name_system": "http://purl.bioontology.org/ontology/LNC/",
"weight_name_identifier": "3141-9"
}

As SDMX:

<Models>
  <Model name="VitalSigns">
  </Model>
</Models>

<Field name="date">2009-05-16T12:00:00Z</Field>
<Field name="encounter">
  <Model name="Encounter">
    <Field name="startDate">2009-05-16T12:00:00Z</Field>
    <Field name="endDate">2009-05-16T16:00:00Z</Field>
    <Field name="facility_name">Wonder Hospital</Field>
    <Field name="facility_adr_country">Australia</Field>
    <Field name="facility_adr_city">WonderCity</Field>
    <Field name="facility_adr_postalcode">5555</Field>
    <Field name="facility_adr_street">111 Lake Drive</Field>
    <Field name="provider_dea_number">325555555</Field>
    <Field name="provider_npi_number">5235235</Field>
    <Field name="provider_email">joshua.mandel@fake.emailserver.com</Field>
    <Field name="provider_name_given">Josuha</Field>
    <Field name="provider_name_family">Mandel</Field>
    <Field name="provider_tel_1_type">w</Field>
    <Field name="provider_tel_1_number">1-235-947-3452</Field>
  </Model>
</Field>
As a Fact object:

```python
from indivo.models import Encounter, VitalSigns
from indivo.lib.iso8601 import parse_utc_date as date

encounter_fact = Encounter(
    startDate=date("2009-05-16T12:00:00Z"),
    endDate=date("2009-05-16T16:00:00Z"),
    facility_name="Wonder Hospital",
    facility_adr_country="Australia",
    facility_adr_city="WonderCity",
    facility_adr_postalcode="5555",
    facility_adr_street="111 Lake Drive",
    provider_dea_number="325555555",
    provider_npi_number="5235235",
    provider_email="joshua.mandel@fake.emailserver.com",
    provider_name_given="Josuha",
    provider_name_family="Mandel",
    provider_tel_1_type="w",
    provider_tel_1_number="1-235-947-3452",
    provider_tel_1_preferred_p=True,
    encounterType_title="Ambulatory encounter",
    encounterType_system="http://smartplatforms.org/terms/codes/EncounterType#",
    encounterType_identifier="ambulatory",
)

encounter_fact.save()

# NOTE: all vitals readings are OPTIONAL. You don't need
# to add all 56 fields here to create a VitalSigns object.

vitals_fact = VitalSigns(
    date=date("2009-05-16T12:00:00Z"),
    encounter=encounter_fact,

    # Blood Pressure
    bp_position_title="Sitting",
    bp_position_identifier="33586001",
    bp_position_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    bp_site_title="Right arm",
    bp_site_identifier="368209003",
    bp_site_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    bp_method_title="Auscultation",
    bp_method_identifier="auscultation",
    bp_method_system="http://smartplatforms.org/terms/codes/BloodPressureMethod#",
    bp_diastolic_unit="mm[Hg]",
    bp_diastolic_value=82,
    bp_diastolic_name_title="Intravascular diastolic",
    bp_diastolic_name_identifier="8462-4",
    bp_diastolic_name_system="http://purl.bioontology.org/ontology/LNC/",
    bp_systolic_unit="mm[Hg]",
    bp_systolic_value=132,
    bp_systolic_name_title="Intravascular systolic",
    bp_systolic_name_identifier="8480-6",
    bp_systolic_name_system="http://purl.bioontology.org/ontology/LNC/",
)
```

# Body Mass Index
bmi_unit="kg/m2",
bmi_value=21.8,
bmi_name_title="Body mass index",
bmi_name_system="http://purl.bioontology.org/ontology/LNC/",
bmi_name_identifier="39156-5",

# Heart Rate
heart_rate_unit="(beats)/min",
heart_rate_value=70,
heart_rate_name_title="Heart rate",
heart_rate_name_system="http://purl.bioontology.org/ontology/LNC/",
heart_rate_name_identifier="8867-4",

# Height
height_unit="m",
height_value=1.8,
height_name_title="Body height",
height_name_system="http://purl.bioontology.org/ontology/LNC/",
height_name_identifier="8302-2",

# Oxygen Saturation
oxygen_saturation_unit="%(HemoglobinSaturation)",
oxygen_saturation_value=99,
oxygen_saturation_name_title="Oxygen saturation",
oxygen_saturation_name_system="http://purl.bioontology.org/ontology/LNC/",
oxygen_saturation_name_identifier="2710-2",

# Respiratory Rate
respiratory_rate_unit="(breaths)/min",
respiratory_rate_value=16,
respiratory_rate_name_title="Respiration rate",
respiratory_rate_name_system="http://purl.bioontology.org/ontology/LNC/",
respiratory_rate_name_identifier="9279-1",

# Temperature
temperature_unit="Cel",
temperature_value=37,
temperature_name_title="Body temperature",
temperature_name_system="http://purl.bioontology.org/ontology/LNC/",
temperature_name_identifier="8310-5",

# Weight
weight_unit="kg",
weight_value=70.8,
weight_name_title="Body weight",
weight_name_system="http://purl.bioontology.org/ontology/LNC/",
weight_name_identifier="3141-9",

Indivo Data Model: Simple_Clinical_Note

Model Definition

As SDML:
As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class SimpleClinicalNote(Fact):
    date_of_visit = models.DateTimeField()
    finalized_at = models.DateTimeField(null=True)
    visit_type = models.CharField(null=True, max_length=100)
    visit_type_type = models.CharField(max_length=80, null=True)
    visit_type_value = models.CharField(max_length=40, null=True)
    visit_type_abbrev = models.CharField(max_length=20, null=True)
    visit_location = models.CharField(max_length=200, null=True)
    specialty = models.CharField(null=True, max_length=100)
    specialty_type = models.CharField(max_length=80, null=True)
    specialty_value = models.CharField(max_length=40, null=True)
    specialty_abbrev = models.CharField(max_length=20, null=True)
    signed_at = models.DateTimeField(null=True)
    provider_name = models.CharField(null=True, max_length=200)
    provider_institution = models.CharField(max_length=200, null=True)
    chief_complaint = models.CharField(null=True, max_length=255)
    content = models.TextField(null=True)
```

Examples

As SDMJ:

```json
{
    "__modelname__": "SimpleClinicalNote",
    "date_of_visit": "2010-02-02T12:00:00Z",
    "finalized_at": "2010-02-03T13:12:00Z",
    "visit_type": "Acute Care",
    "visit_type_type": "http://codes.indivo.org/visit-types#",
    "visit_type_value": "123",
    "visit_type_abbrev": "acute",
    "visit_location": "Longfellow Medical",
    "specialty": "Hematology/Oncology",
    "content": "String"
}
```
"specialty_type": "http://codes.indivo.org/specialties#",
"specialty_value": "234",
"specialty_abbrev": "hem-onc",
"signed_at": "2010-02-03T12:00Z",
"provider_name": "Kenneth Mandl",
"provider_institution": "Children’s Hospital Boston",
"chief_complaint": "stomach ache",
"content": "Patient presents with ..."
}

As SDMX:

<Models>
  <Model name="SimpleClinicalNote">
    <Field name="date_of_visit">2010-02-02T12:00:00Z</Field>
    <Field name="finalized_at">2010-02-03T13:12:00Z</Field>
    <Field name="visit_type">Acute Care</Field>
    <Field name="visit_type_type">http://codes.indivo.org/visit-types#</Field>
    <Field name="visit_type_value">123</Field>
    <Field name="visit_type_abbrev">acute</Field>
    <Field name="visit_location">Longfellow Medical</Field>
    <Field name="specialty">Hematology/Oncology</Field>
    <Field name="specialty_type">http://codes.indivo.org/specialties#</Field>
    <Field name="specialty_value">234</Field>
    <Field name="specialty_abbrev">hem-onc</Field>
    <Field name="signed_at">2010-02-03T13:12:00Z</Field>
    <Field name="provider_name">Kenneth Mandl</Field>
    <Field name="provider_institution">Children’s Hospital Boston</Field>
    <Field name="chief_complaint">stomach ache</Field>
    <Field name="content">Patient presents with ...</Field>
  </Model>
</Models>

As a Fact object:

```python
from indivo.models import SimpleClinicalNote
from indivo.lib.iso8601 import parse_utc_date as date

simple_clinical_note_fact = SimpleClinicalNote(
    date_of_visit=date("2010-02-02T12:00:00Z"),
    finalized_at=date("2010-02-03T13:12:00Z"),
    visit_type="Acute Care",
    visit_type_type="http://codes.indivo.org/visit-types#",
    visit_type_value="123",
    visit_type_abbrev="acute",
    visit_location="Longfellow Medical",
    specialty="Hematology/Oncology",
    specialty_type="http://codes.indivo.org/specialties#",
    specialty_value="234",
    specialty_abbrev="hem-onc",
    signed_at=date("2010-02-03T13:12:00Z"),
    provider_name="Kenneth Mandl",
    provider_institution="Children’s Hospital Boston",
    chief_complaint="stomach ache",
    content="Patient presents with ..."
)
```

4.5. Indivo Data Models 355
4.5.5 Advanced Data-Model Tasks

Adding Advanced Features to a Data-Model

For complicated data models, a simple SDML definition just won’t suffice. For a few specific features, such as custom object serialization or creation-time field validation, you can define (in python) an extra options file for a data model.

This file should be named extra.py, and can be dropped into the filesystem next to any data model, as described below. The file should contain subclasses of indivo.data_models.options.DataModelOptions, each of which describes the options for one data model defined in the model.py file in the same directory. Options are:

```python
class indivo.data_models.options.DataModelOptions(Type)
    Defines optional extra functionality for Indivo datamodels.
```

To add options to a datamodel, subclass this class and override its attributes.

Currently available options are:

- `model_class_name`: Required. The name of the datamodel class to attach to.
- `serializers`: Custom serializers for the data model. Should be set to a subclass of indivo.serializers.DataModelSerializers.
- `field_validators`: Custom validators for fields on the data model. A dictionary, where keys are field names on the model, and values are lists of Django Validators to be run against the field.

For example, here’s our options file for the Problem data model:

```python
from indivo.serializers import DataModelSerializers
from indivo.data_models.options import DataModelOptions
from indivo.validators import ExactValueValidator

SNOMED_URI = 'http://purl.bioontology.org/ontology/SNOMEDCT/'

class ProblemSerializers(DataModelSerializers):
    def to_rdf(queryset, result_count, record=None, carenet=None):
        # ... our SMART RDF serializer implementation here ... #
        return 'some RDF'

class ProblemOptions(DataModelOptions):
    model_class_name = 'Problem'
    serializers = ProblemSerializers
    field_validators = {
        'name_system': [ExactValueValidator(SNOMED_URI)],
    }
```

Make sure to restart Indivo for your changes to take effect after you add your extra.py file—but there’s no need to reset Indivo.

Adding Custom Serializers to a Data-Model

By default, when returning data via the generic reporting API, Indivo will attempt to serialize data as SDMJ or SDMX, depending on the requested response format. If you need your data to come back in other formats, or if the default serializers aren’t smart enough to represent your data model correctly, you can implement custom serializers for the data model.
Defining the Serializers

Serializers for a data model are implemented as simple methods that take a Django queryset object, and return a serialized string. For a given data-model, you should define a subclass of `indivo.serializers.DataModelSerializers`, and add your desired serializers as methods on the class. Currently, available serializers are:

**to_xml** *(queryset, result_count, record=None, carenet=None)*

returns an XML string representing the model objects in `queryset`.

Parameters

- **queryset** *(QuerySet)* – the objects to serialize
- **result_count** *(integer)* – the total number of items in `queryset`
- **record** *(Record)* – the patient record that the objects belong to, if available.
- **carenet** *(Carenet)* – the Carenet via which the objects have been retrieved, if available.

Return type  string

**to_json** *(queryset, result_count, record=None, carenet=None)*

returns a JSON string representing the model objects in `queryset`.

Parameters

- **queryset** *(QuerySet)* – the objects to serialize
- **result_count** *(integer)* – the total number of items in `queryset`
- **record** *(Record)* – the patient record that the objects belong to, if available.
- **carenet** *(Carenet)* – the Carenet via which the objects have been retrieved, if available.

Return type  string

**to_rdf** *(queryset, result_count, record=None, carenet=None)*

returns an RDF/XML string representing the model objects in `queryset`.

Parameters

- **queryset** *(QuerySet)* – the objects to serialize
- **result_count** *(integer)* – the total number of items in `queryset`
- **record** *(Record)* – the patient record that the objects belong to, if available.
- **carenet** *(Carenet)* – the Carenet via which the objects have been retrieved, if available.

Return type  string

For example, here’s a (non-functional) implementation of the serializers for the Problems data-model:

```python
from indivo.serializers import DataModelSerializers

class ProblemSerializers(DataModelSerializers):
    def to_xml(self, queryset, result_count, record=None, carenet=None):
        return '''<Problems>...bunch of problems here...</Problems>'''

    def to_json(self, queryset, result_count, record=None, carenet=None):
        return '''[["Problem": "data here"], ["Problem": "More data here..."]]'''

    def to_rdf(self, queryset, result_count, record=None, carenet=None):
        return '''<rdf:RDF><rdf:Description rdf:type='indivo:Problem'>...RDF data here...</rdf:Description></rdf:RDF>'''
```

4.5. Indivo Data Models
A couple things to note:

- The `to_*()` methods DO NOT take `self` as their first argument. Under the hood, we actually rip the methods out of the serializers class and attach them directly to the data-model class.
- The `model_class_name` attribute is required, and indicates which data-model the serializers should be attached to.

### Libraries for Serialization

When serializing models, the following libraries can come in handy:

- **lxml.etree**: Our favorite XML manipulation library. See [http://lxml.de/tutorial.html](http://lxml.de/tutorial.html) for the details. Lxml is required for a running Indivo instance, so it will always be available for import (`from lxml import etree`).


### Attaching the Serializers to a Data Model

Adding custom serializers to a data-model is simple: simply set your `DataModelSerializers` subclass to the `serializers` attribute of a `DataModelOptions` subclass in an `extra.py` file (see above for info on adding advanced data-model options).

### Adding Field Validation to a Data-Model

By default, data models defined in SDML are very permissive: all fields are nullable, and there are no constraints on valid data points other than their type (string, date, etc.). In some cases, a data element could satisfy these constraints, but still be invalid. For example, an Indivo Problem must have its name coded using SNOMED, so a problem without a snomed code is invalid.

### Defining the Validators

In such cases, you can attach validators to the data model. Django Validators are essentially just python callables that raise a `django.core.exceptions.ValidationError` if they are called on an invalid data point. We’ve defined a couple of useful validators, though you could use any function you’d like.

For example, here’s a validator that will accept only the value 2:

```python
from django.core.exceptions import ValidationError
def validate_2(value):
    if value != 2:
        raise ValidationError("Invalid value: %s. Expected 2\n%str(value))
```
### Built in Validators

Django provides a number of built-in validators, for which a full reference exists here: https://docs.djangoproject.com/en/1.2/ref/validators/#built-in-validators.

In addition, Indivo defines a few useful validators in `indivo.validators`:

```python
class indivo.validators.ValueInSetValidator(valid_values, nullable=False)
    Validates that a value is within a set of possible values.
    
    The optional ‘nullable’ flag determines whether or not the value may also be empty.

class indivo.validators.ExactValueValidator(valid_value, nullable=False)
    Validates that a value is exactly equal to a certain value.
    
    The optional ‘nullable’ flag determines whether or not the value may also be empty.
```

### Attaching Validators to a Data Model

Adding custom validators to a data-model is simple: simply add the validator to the field_validators attribute of a `DataModelOptions` subclass in an `extra.py` file (see above for info on adding advanced data-model options).

For example, let’s add the requirement that Problem names must be coded as snomed. We can write the validator using the built-in `ExactValueValidator`:

```python
from indivo.validators import ExactValueValidator
SNOMED_URI = 'http://purl.bioontology.org/ontology/SNOMEDCT/
snomed_validator = ExactValueValidator(SNOMED_URI)
```

We can then attach it to the `name_system` field of a Problem, which will guarantee that we only accept problems which identify themselves as having a snomed code for their names:

```python
class ProblemOptions(DataModelOptions):
    model_class_name = 'Problem'
    field_validators = {
        'name_system': [snomed_validator]
    }
```

Note that we put `snomed_validator` in a list, since we might theoretically add additional validators to the `name_system` field.

### Adding Custom Data-Models to Indivo

As of version 1.1 of Indivo X, we’ve added a feature that makes it much easier to add (in a drag-and-drop fashion) new supported data models to an instance of Indivo. Adding a new data model to Indivo involves:

- Creating the data model definition
- Dropping the data model into the filesystem
- Migrating the database tables to support the new model

#### Defining the Data Model

As you saw above, data models can be defined in two formats: SDML or Django model classes. Simply produce a definition in one of the two forms, and save it to a file named `model.sdml` or `model.py`.
Dropping the Definition into the Filesystem

Indivo data models currently have the following layout on the filesystem:

```plaintext
indivo_server/
  indivo/
    ...
    data_models/
      core/
        allergy/
          model.[sdml | py]
          example.[sdmj | sdmx | py]
          extra.py
        ...
      contrib/
```

The `indivo/data_models/core/` directory contains all of our built-in data models, and you shouldn’t modify it. Since you are ‘contributing’ a data model to Indivo, add your data model to the `indivo/data_models/contrib/` directory. Simply:

- **Create a new subdirectory** under `indivo/data_models/contrib/`.
- **Drop your model definition** into that directory. This file MUST BE NAMED MODEL.PY OR MODEL.SDML to be identified as a data model.
- **Add (optional) example files** into that directory. Files should be named `example.sd mj`, `example.sdmx`, or `example.py`, and should be example instances of the data model as SDMJ, SDMX, or Fact objects respectively. If present, they will help others use and document your data model.
- **Add an (optional) extras file** to the directory. The file must be named `extra.py`, and may contain extra options for your data-model, such as custom serializers.
- **Your final directory structure** should now look something like:

```plaintext
indivo_server/
  indivo/
    ...
    data_models/
      core/
        allergy/
          model.[sdml | py]
          example.[sdmj | sdmx | py]
          extra.py
        ...
      contrib/
        your_data_model/
          model.[sdml | py]
          example.[sdmj | sdmx | py]
          extra.py
```

Migrating the Database

Indivo relies on the South migration tool to get the database synced with the latest data models. Once you’ve dropped your data model into the filesystem, South should be able to detect the necessary changes.

To detect the new model and generate migrations for it, run (from the `indivo_server` directory):

```bash
python manage.py schemamigration indivo --auto
```
You should see output like:

+ Added model indivo.YOURMODELNAME
Created 0018_auto__add_model_YOURMODELNAME.py. You can now apply this migration with: ./manage.py migrate indivo

To do a quick sanity check that you aren’t about to blow away your database, run:

`python manage.py migrate indivo --db-dry-run -v2`

This should output the SQL that will be run. Make sure this looks reasonable, ESPECIALLY if you are running Indivo on Oracle, where the South tool is still in alpha. If the SQL looks reasonable, go ahead and run the migration, with:

`python manage.py migrate indivo`

And you’re all set!

Next Steps

Make sure to restart Indivo for your changes to take effect.

See also:

Now you’ve added a new data model to Indivo: Congratulations! It can be stored in the database and queried via the API.

But until you map a Schema to it, you won’t be able to actually add data to your new model. To learn more, see:

- Data Pipeline
- Adding Custom Schemas to Indivo
- Adding Custom Transforms to Indivo

4.6 Indivo Simple Data Modeling Language (SDML)

4.6.1 Introduction

As of version 1.1, Indivo supports drag-and-drop substitution of data models within the filesystem. We expect that this will encourage Indivo administrators to try their hands at building new data models for use with Indivo, but we recognize that many will not have the python expertise necessary to implement models using the Django web framework. To enable the rapid development and deployment of very simple data models without the overhead of learning Django, we have defined a basic language for representing data models: Simple Data Modeling Language, or SDML.

Data in Indivo is written to the database as python objects using Django’s Object-Relational Mapper: this will be equally problematic for those without python experience. Therefore, we have also defined languages for representing data in JSON or XML with similar degrees of simplicity: Simple Data Modeling JSON (SDMJ) and Simple Data Modeling XML (SDMX).

See also:

This document contains the specifications for these languages. For information on how and when to use them, see:
4.6.2 Defining Data Models: SDML

There are a number of existing languages for defining data models (see http://en.wikipedia.org/wiki/Category:Data_modeling_languages for a listing). However, most of these are much too generalized and powerful for our use case. What we want is a language that is very simple to use (low learning curve), and very limited in functionality (medical data models shouldn’t be too complex, and shouldn’t need most of the capabilities of a relational database). SDML therefore uses JSON’s syntax exactly, but has restrictions that allow us to define Django models with it.

Syntax

- A data model in SDML is represented by a JSON object: `{}`
- Attributes are represented by JSON pairs: `'name': 'value'`
- All attributes are optional.
- The special attribute named '__modelname__' defines the name of the model.
- Fieldtypes (which are defined by the ‘value’ of an attribute) are restricted to the following:
  - Number
  - String
  - Date
  - CodedValue
  - ValueAndUnit
  - Address
  - Name
  - Telephone
  - Pharmacy
  - Provider
  - VitalSign
  - BloodPressure
  - ValueRange
  - QuantitativeResult
  - One-to-One
  - One-to-Many
- All types but One-to-One and One-to-Many are indicated by a simple string (i.e. ‘Number’ | ‘CodedValue’ | ‘Provider’)
• One-to-One fields are indicated by a sub-object, and may be nested arbitrarily:

```
{  
    "__modelname__": "mymodel",
    "field1": "Date",
    "field2": {
        "__modelname__": "mysubmodel",
        "subfield1": "String",
        "subfield2": "Number",
        "subfield3": {
            "__modelname__": "mysubsubmodel",
            "subsubfield1": "String"
        }
    }
}
```

• One-to-Many fields are indicated by a list containing a definition of a sub-object:

```
{  
    "__modelname__": "mymodel",
    "field1": "Date",
    "field2": [{
        "__modelname__": "mysubmodel",
        "subfield1": "String",
        "subfield2": "Number"
    }]
}
```

And that's it.

**Example**

Here is an example definition of a medication data model (more complicated than our model, actually) using SDML:

```
{  
    "__modelname__": "TestMedication",
    "name": "String",
    "date_started": "Date",
    "date_stopped": "Date",
    "brand_name": "String",
    "route": "String",
    "prescription": {
        "__modelname__": "TestPrescription",
        "prescribed_by_name": "String",
        "prescribed_by_institution": "String",
        "prescribed_on": "Date",
        "prescribed_stop_on": "Date"
    },
    "fills": [{
        "__modelname__": "TestFill",
        "date_filled": "Date",
        "supply_days": "Number",
        "filled_at_name": "String"
    }]
}
```

This definition will create three new data models: `TestMedication`, `TestPrescription`, and `TestFill`. It will add a one-to-one relation between TestMedication and TestPrescription, and a one-to-many relation between TestMedication and TestFill. That is to say, each TestMedication might have one prescription and multiple fills.
4.6.3 Representing Data: SDMJ and SDMX

Simple Data Modeling JSON (SDMJ) and Simple Data Modeling XML (SDMX) are two nearly identical methods of representing data that matches an SDML definition. The only difference is the form of the envelope around the data.

SDMJ

SDMJ looks exactly like SDML, with four key differences:

- The datatypes in SDML (‘Number’, ‘String’, ‘Date’) are replaced by the datapoints in SDMJ
- Each model has an attribute ‘__documentid__', specifying the ID of the source document
- Since all attributes are optional, any attribute may be omitted in SDMJ.
- In one-to-many attributes, SDMJ actually specifies multiple datapoints, instead of just defining a submodel in a list.

Here’s an example SDMJ document matching the SDML definition above:

```json
{
    "__modelname__": "TestMedication",
    "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
    "name": "ibuprofen",
    "date_started": "2010-10-01T00:00:00Z",
    "date_stopped": "2010-10-31T00:00:00Z",
    "brand_name": "Advil",
    "prescription": {
        "__modelname__": "TestPrescription",
        "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
        "prescribed_by_name": "Kenneth D. Mandl",
        "prescribed_by_institution": "Children’s Hospital Boston",
        "prescribed_on": "2010-09-30T00:00:00Z",
        "prescribed_stop_on": "2010-10-31T00:00:00Z"
    },
    "fills": [
        {
            "__modelname__": "TestFill",
            "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
            "date_filled": "2010-10-01T00:00:00Z",
            "supply_days": "15",
            "filled_at_name": "CVS"
        },
        {
            "__modelname__": "TestFill",
            "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
            "date_filled": "2010-10-16T00:00:00Z",
            "supply_days": "15",
            "filled_at_name": "CVS"
        }
    ]
}
```

Note: we’ve removed the ‘route’ attribute as it was not required, and have added two fills. This will result in 4 Fact objects being saved to the database: one TestMedication, one TestPrescription, and two TestFills.
SDMX

SDMX looks exactly like SDMJ, with the exceptions that:

- It’s XML
- attribute-value pairs are represented as `<Field name="attribute_name">attribute_value</Field>`
- The `__modelname__` attribute is pulled out as a toplevel tag with `documentId` as an attribute: `<Model name="model_name" documentId="id">`
- Each `<Model />` tag contains an attribute `documentId`, specifying the ID of the source document
- In order to represent multiple toplevel datapoints, SDMX must always live under a root `<Models>` tag.

Here’s the same example document we just saw as SDMJ in SDMX form:

```xml
<Models>
  <Model name="TestMedication" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
    <Field name="date_started">2010-10-01T00:00:00Z</Field>
    <Field name="name">ibuprofen</Field>
    <Field name="brand_name">Advil</Field>
    <Field name="date_stopped">2010-10-31T00:00:00Z</Field>
    <Field name="prescription">
      <Model name="TestPrescription" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
        <Field name="prescribed_by_name">Kenneth D. Mandl</Field>
        <Field name="prescribed_by_institution">Children’s Hospital Boston</Field>
        <Field name="prescribed_on">2010-09-30T00:00:00Z</Field>
        <Field name="prescribed_stop_on">2010-10-31T00:00:00Z</Field>
      </Model>
    </Field>
    <Field name="fills">
      <Models>
        <Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
          <Field name="date_filled">2010-10-01T00:00:00Z</Field>
          <Field name="supply_days">15</Field>
          <Field name="filled_at_name">CVS</Field>
        </Model>
        <Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
          <Field name="date_filled">2010-10-16T00:00:00Z</Field>
          <Field name="supply_days">15</Field>
          <Field name="filled_at_name">CVS</Field>
        </Model>
      </Models>
    </Field>
  </Model>
</Models>
```

For those who like to work with XML in conjunction with schemas, here’s an XSD which describes SDMX and can be used to validate it:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:complexType name="ModelType">
    <xs:sequence>
      <xs:element name="Field" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType mixed="true">
          <xs:choice>
            <xs:element name="Model" type="ModelType" minOccurs="0" maxOccurs="1"/>
            <xs:element name="Models" type="ModelsType" minOccurs="0" maxOccurs="1"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
Representing Dates

Strings and Numbers in SDMX and SDMJ should simply be input as JSON or XML literals as appropriate. Dates work the same way, with the exception that they need to be formatted (as with all dates and date-times in Indivo) as ISO 8601 UTC timestamps as described in Indivo Basic Data Formats.

4.7 Data Pipeline

4.7.1 Introduction

The Indivo Data Pipeline is the set of processes that take input data as it enters Indivo, extract and store clinical datapoints (Facts), and make those datapoints available as output via the API. As of version 1.1 of Indivo X, every component of the pipeline is substitutable: you can add new formats for input data, new ways to store the data, and new methods to output them.

Let’s start with some vocabulary:

Schema A description of a format for data, which can also be used to validate that data. For example, our SDMX Schema defines the format of incoming data in our SDMX specification language, and can be used to determine whether an XML document is valid SDMX. In Indivo, we use schemas to define the formats in which we accept data via the API, and validate that data as it comes in. Right now, schemas can only take the form of XSDs, since we accept input data only in XML form, but in the future this might change. More on Indivo schemas here.

Document A collection of data, formatted according to a Schema. When you use our API to add data to Indivo, the data you send to Indivo constitutes one document.

Transform A tool that takes as input some data which validates against a schema, and outputs it in a format consistent with Indivo’s Data Models. We currently accept two types of transforms: XSLTs and Python code. We also understand several output formats for the data after it has been transformed. More on transforms in Indivo here.

Data Model A definition of the format in which processed data is stored in Indivo’s database. Each ‘data model’ corresponds to a type of clinical information. For example, our Medication Data Model describes how we represent a processed medication in the database. We currently accept two types of data model definitions. More on Indivo Data Models here.
Fact  A single datapoint: an instance of a data model. For example, one Medication is a Fact, as is one Problem, etc. The medication fact is an object whose format is defined by the Medication Data Model. When you take input data in the form of an XML Document and run it through a Transform, you arrive at a Fact object that can be stored in the database.

Serialization  The process of taking a Fact object and converting it to an output format, suitable for returning in response to an API call. For example, our current Reporting API outputs Fact objects serialized to XML or JSON.

4.7.2 The Pipeline

As you may have surmised from the above definitions, the data pipeline in Indivo is actually pretty simple, and consists of five steps:

1. Identification. An incoming document is examined, and its type determined (right now, since data comes only in XML documents, a document’s type can be uniquely constructed from it’s root nodename and namespace, i.e. http://indivo.org/vocab/xml/documents#Allergy).

2. Validation. The identified document is matched against its schema. If it is invalid, the pipeline terminates, and an error is thrown.

3. Transformation. The validated document is then processed using its transform. If the output matches one of the valid transform output formats, it is converted into one or more Fact objects, ready for storage.

4. Storage. The processed Facts are written to the database.

5. Retrieval. When an app makes an API call using the Reporting API, the database is queried for matching Facts. Those facts are then serialized into the required output format (i.e., XML or JSON) and returned to the app.

With the new data pipeline, Schemas, Transforms, and Data Models are all substitutable: you can add and replace them at will.

Notice that there is not a one-to-one relationship between incoming documents and processed fact objects. This allows for Indivo to accept schemas like a CCR, which contains many facts. A parsed CCR document might end up outputting many Problem, Medication, Allergy, Lab, or other facts, even though there was only one input document.

Conversely, there is not one document type associated with producing one type of fact object. This allows Indivo to accept the same type of data in many formats. For example, you could get a medication fact from our standard medication document, but you could also get a medication fact from a CCR.

The data pipeline is activated whenever new data is added to Indivo using document creation API calls, and whenever data is retrieved from Indivo using the reporting API calls. The following calls add new documents to Indivo, and therefore feed data into the pipeline:

- POST /records/{RECORD_ID}/documents/
- PUT /records/{RECORD_ID}/documents/external/{APP_ID}/{EXTERNAL_ID}
- POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace
- PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace/external/{APP_ID}/{EXTERNAL_ID}
- POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/
- PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/external/{APP_ID}/{EXTERNAL_ID}
- POST /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}/accept

All of the reporting calls retrieve collections of processed Fact objects from Indivo, and thus rely on the tail end of the processing pipeline.
Figure 4.1: The Indivo Data Pipeline, Visualized
4.7.3 Learning More

See also:

More information on the components of the data pipeline can be found here:

- Indivo Schemas
- Indivo Transforms
- Indivo Data Models

More information on customizing the data pipeline can be found here:

- Adding Custom Schemas to Indivo
- Adding Custom Transforms to Indivo
- Adding Custom Data-Models to Indivo

4.8 Indivo Localization

Indivo is developed entirely in America, and all of its text is written in English. However, the value of a PCHR should not be limited to those who can read a specific language, and all of Indivo’s core functionality is not language-specific. Indivo therefore provides functionality for Localization (abbreviated L10n), or adapting the software for use in a specific region by translating its text. Indivo builds upon Django’s native support for L10n in order to accomplish this.

4.8.1 Overview

There are four main steps involved in localizing Indivo: marking strings for translation, creating language files for a specific locale, translating the strings, and compiling the translated strings. This document will cover each of these steps. Note that all Indivo Localization is implemented in our reference UI Server, as Indivo Server itself produces pure XML, not human-readable text. Any Indivo UI app written in Django could use the same steps to achieve localization, however.

4.8.2 Supported Locales

You may not have to go to the trouble to do any translation. Below is a list of currently supported languages that Indivo has already been translated into:

- en_US: American English

If you found your language on the above list, simply edit your settings.py file as described below and Indivo will be automatically translated for you.

4.8.3 Marking Strings for Translation

In order to translate Indivo, all human-readable text produced by the UI must be identified for translation. We’ve taken a first stab and identified most of the strings that we think need to be translated, but we may well have missed some. If you find an untranslated string in the source code that needs translation, you can mark it as follows:

- Surround the text with `{% trans %}` tags. For example,
<h>ENGLISH HEADER</h>

becomes

<h>{% trans "ENGLISH HEADER" %}</h>

• If the text is a block paragraph, or contains other Django template tags, use the {{ blocktrans }} tag:

<nowiki><p>Now I’m talking about something important! And refering to the variable {{var}}!</p></nowiki>

becomes

<nowiki><p>{{ blocktrans %}Now I’m talking about something important! And refering to the variable {{var}}!{% endblocktrans %}</p></nowiki>

• Note: Indivo UI Server produces text in .html files, .ejs files, and .js files. We’ve added some code to allow you to treat all of these files the same way: simply wrap the string you want translated in the appropriate Django tags.

• For more detailed information, see Django’s documentation on this here.

• Tell us about your changes! Submit a github pull request, and we will incorporate your tagged strings so that others can translate them into all of the Indivo supported locales.

4.8.4 Creating Message Files

Generate the File

Once you’ve indicated which strings should be translated, Django will create a single file containing all of the strings you need to translate in order to localize Indivo. This file, called a ‘message file’, has the extension .po and can be created by running (from the top-level indivo_ui_server directory):

```
python manage.py makemessages -l LOCALE -e .html,.ejs,.py
```

Where LOCALE refers to the language code you wish to translate to, indicated in what Django calls ‘locale name format’. Basically, this means that the language code takes the form ll[_{CC}], where the first two characters, lowercase, represent the desired language, and the optional last two characters, uppercase, represent the country variant of the language. For example, it for italian, and pt_BR for Brazilian portuguese. For a complete listing of the locales that Django current supports, see the listing in their source code.

A Look at the File

If you open your *.po file, which was placed in indivo_ui_server/locale/LOCALE/LC_MESSAGES/, you will see, for each marked string, the following structure:

```
#: path/to/python/module.py:23
msgid "Welcome to my site."
msgstr ""
```

The line beginning with # indicates the source file where the message came from, the msgid line is the original English text to be translated, and the msgstr line is where you will write your translations.

4.8.5 Translating Strings

Unfortunately, Django isn’t smart enough to actually do all of the translation for you (and if it did, Indivo would come out looking like it had been run through Google translate). To get an accurate translation of Indivo into your local
language, you will need to go through the .po file generated in the previous section and translate each string indicated by a line beginning with msgid. For each such string, translate the string into your local language, and place the result in the line just below it (starting with msgstr). This is the tedious part of localization, but it is a necessary one to insure a complete translation.

4.8.6 Compiling Message Files

Now that you’ve translated Indivo, you need to get those translations into a pre-compiled format so that Django can be efficient in translating Indivo in realtime. This is done with a simple command:

```python
python manage.py compilemessages
```

This compiles your *.po file into a *.mo file, which Django uses when Indivo is running.

**Note:** If you ever go back and change any of your translated strings in the *.po file, you’ll need to recompile the file into a *.mo file. Likewise, if you mark additional strings for translation, you’ll need to regenerate the message file for the new strings to be available for translation.

4.8.7 Wrapping Up

And that’s it! You’ve successfully translated Indivo. A few final things to consider:

- If you ever go back and change any of your translated strings in the *.po file, you’ll need to recompile the file into a *.mo file. Likewise, if you mark additional strings for translation, you’ll need to regenerate the message file for the new strings to be available for translation.

- You still need tell Indivo to translate into your particular language: you can do this by editing `indivo_ui_server/settings.py` and changing the `LANGUAGE_CODE` setting to match your current locale. Note that in `settings.py`, the `LANGUAGE_CODE` should be of the form `ll[-cc]`, not `ll[_CC]` (note the dash separator and lowercase country code).

- If you’ve translated Indivo into a new language, we want to hear about it! Submit your translation to us as a pull request on github, and we’ll make sure that others have access to it.

- For a more detailed description of how Django handles L10n, see [https://docs.djangoproject.com/en/1.2/topics/i18n/localization/](https://docs.djangoproject.com/en/1.2/topics/i18n/localization/).

4.9 Registering Apps with Indivo

As of version 2.0, Indivo has an official process for adding and managing apps on a given instance. For those of you familiar with using `indivo_data.xml` for adding apps, the `<machine_apps>` and `<user_apps>` tags will no longer be respected, and will not add apps to Indivo.

4.9.1 App Manifests

Previous releases of Indivo have relied on an implicit XML syntax in `indivo_data.xml` for representing an app within Indivo. Now, applications must provide a declared manifest describing the application and its requirements. We have adopted the SMART Project’s syntax for describing apps in manifests, but have added some Indivo-specific parameters to accommodate the way Indivo represents applications. For compatibility, any valid SMART manifest will also be a valid Indivo manifest.

See The SMART Project’s documentation for a description of the basic syntax of a manifest, which is JSON based. Below, we describe only the Indivo-specific modifications.
User Apps

New Fields

An Indivo user-app may define (beyond the SMART-supported manifest fields) any of the following additional properties in its manifest:

- **oauth_callback_url**: A callback URL for Indivo-style OAuth access
- **autonomous_reason**: An explanation for why the app requires offline access to patient records
- **has_ui**: `true` or `false`, whether the app can be displayed in a browser.
- **frameable**: `true` or `false`, whether the app should be loaded in an iframe in the Indivo UI.
- **indivo_version**: Required version of Indivo for compatibility

Changes from the indivo_data.xml fields

The following fields have changed names to match the SMART manifest fields:

- **email**: Has been renamed `id`, per SMART.
- **start_url_template**: Has been renamed `index`. If using Indivo-style OAuth authentication, the same templating parameters may be passed in the URL (i.e. `{record_id}`)
- **is_autonomous**: Has been moved to the SMART `mode` property. Acceptable modes are:
  - **background**: This app will act like an autonomous Indivo app.
  - **ui**: This app will act like a non-autonomous Indivo app.

Example

```json
{
    "name": "Problems",
    "description": "Display a list of problems, or enter new ones.",
    "author": "Arjun Sanyal, Children’s Hospital Boston",
    "id": "problems@apps.indivo.org",
    "version": "1.0.0",
    "smart_version": "0.4",

    "mode": "ui",
    "scope": "record",
    "has_ui": true,
    "frameable": true,

    "icon": "jmvc/ui/resources/images/app_icons_32/problems.png",
    "index": "/apps/problems/start_auth?record_id={record_id}&carenet_id={carenet_id}"
    "oauth_callback_url": "/apps/problems/after_auth"
}
```
UI and Admin Apps

New Fields

An Indivo user-app may define (beyond the SMART-supported manifest fields) any of the following additional properties in its manifest:

- `ui_app`: true or false. Whether the machine app is a UIApp ('chrome app').
- `indivo_version`: Required version of Indivo for compatibility

Changes from the `indivo_data.xml` fields

The following fields have changed names to match the SMART manifest fields:

- `email`: Has been renamed `id`, per SMART.
- `app_type`: Please use the `ui_app` field, above.

Example

```json
{
    "name": "Sample UI App",
    "description": "The reference Indivo UI App",
    "author": "Ben Adida, Travers Francke, Arjun Sanyal, Pascal Pfiffner, Daniel Haas. Children’s Hospital Boston",
    "id": "chrome@apps.indivo.org",
    "version": "2.0.0",
    "indivo_version": "2.0.0",
    "ui_app": true
}
```

4.9.2 App OAuth Credentials

When authenticating to Indivo using traditional OAuth, applications must provided Indivo with their consumer key and a shared consumer_secret. As this secret is private and should not be shared with other apps (i.e., via a call to `GET /apps/`), it should be registered in a separate file. We therefore define a simple JSON format for specifying app OAuth credentials, which is simple JSON and has two fields:

- `consumer_key`: The OAuth consumer key for the app.
- `consumer_secret`: The OAuth consumer secret for the app.

Here’s a sample credentials file, for our built-in Problems app:

```json
{
    "consumer_key": "problems@apps.indivo.org",
    "consumer_secret": "SECRTFORTHEPROBLEMSAPP:CHANGEME"
}
```

Notes:

- If your app is a SMART app, you probably haven’t explicitly generated a ‘consumer key’. You should set the `consumer_key` field of the credentials file to match the `id` field of your app manifest file.
- If your app is a SMART CONNECT app (or connects to Indivo using In-Browser Connect Authentication), you do not need a consumer secret. In such a case, set the `consumer_secret` field of the credentials file to the empty string: ‘’.
4.9.3 Managing the Registered Apps

Thus, to register an app with Indivo, you need two files: an app manifest (manifest.json) and a credentials file (credentials.json).

Changing the set of registered apps in Indivo is now drag-and-drop, as with our process for managing datatypes and schemas. To add, remove, or change an app, you’ll need to:

- Create a manifest and credentials file for the app (or modify existing manifests/credentials)
- Drop the files into the filesystem
- Sync the database with the filesystem

Apps in the Indivo Filesystem

Indivo apps currently have the following layout on the filesystem:

```
indivo_server/
    registered_apps/
        admin/
        ui/
        user/
            allergies/
                manifest.json
                credentials.json
            ...
```

To add an app to the filesystem, simply add a subdirectory under `indivo_server/registered_apps/admin`, `indivo_server/registered_apps/ui`, or `indivo_server/registered_apps/user` (depending on the type of your app), and drop a manifest and a credentials file into that directory.

To remove an app, just delete its directory.

To change an app’s manifest or credentials, just modify the appropriate `manifest.json` or `credentials.json` file.

Syncing the Database with the Filesystem

To alert Indivo that you’ve changed the registered apps, run (from `indivo_server/`):

```
python manage.py sync_apps
```

This will process the list of registered apps and sync any additions, deletes or updates to the database.

Resetting Indivo

With the new system, there is **NO NEED TO RESET INDIVO TO ADD APPS!**. Simply run the `sync_apps` command, above.

When you do reset Indivo, the reset script now calls `sync_apps`, which will add all of the registered apps to Indivo.
4.10 Sample Data in Indivo

Testing out new Indivo apps is difficult at best with no patient data to run against. That’s why we’ve added the capability to create records with sample patient data pre-loaded into them. There are two pathways for getting sample data into Indivo: using the indivo_data.xml file and the reset.py script, and putting Indivo into Demo Mode.

4.10.1 Using indivo_data.xml

The simplest way to get sample data into Indivo is to use the basic reset script. When you’re setting up indivo_data.xml, just add a data_profile attribute to any <record> tags that you’re creating. The default indivo_data.xml file has an example of this already: it loads the data profile for patient_2 into the John S. Smith record.

As described below, you can determine available data profiles by looking at the subdirectories of settings.SAMPLE_DATA_DIR.

4.10.2 Using Demo Mode

If you would like to run Indivo fully populated by sample data (as we do on our developer’s sandbox), you can put Indivo into Demo Mode. In this mode, all newly created accounts are immediately set up with records pre-loaded with sample data.

You can do this by configuring the following settings in settings.py:

```
SAMPLE_DATA_DIR The directory where sample data is located.
DEMO_MODE Puts Indivo into Demo Mode if set to True.
DEMO_PROFILES A dictionary mapping record labels to data profiles to load for each new account. For example, if the value of this settings were:

{'John Doe': 'patient_1',
 'Robert Frost': 'bob',
 'Ted Kennedy': 'patient_2',
}

Then for each new account, three new records would be created. The first would have a label of ‘John Doe’, and be populated by the data profile ‘patient_1’. The second would have a label of ‘Robert Frost’, and be populated by the data profile ‘bob’. The third would have a label of ‘Ted Kennedy’, and be populated by the data profile ‘patient_2’.

Note: Demo Mode autocreates records any time the API call to create an account, POST /accounts/, is called. This means that any records created through other means (i.e. by a call to POST /records/) will not be populated with data. If your registration UI or admin app handles record creation, this could lead to the existence of some records populated with sample data, and others without it.

4.10.3 Available Sample Data

Any data in the directory specified by settings.SAMPLE_DATA_DIR (settings.APP_HOME/sample_data by default) is available for loading into Indivo. Data in SAMPLE_DATA_DIR should look like:

```
profile_1/  # Data profiles. Each directory should correspond to a single patient.
bob/       # For example, this is the data profile that can be referenced as 'bob'
```

4.10. Sample Data in Indivo
Namely, the data directory should have multiple subdirectories, each representing one patient’s data. Within a patient’s directory, there might be a `Demographics.xml` file. There will also be any number of data files, labeled `doc_{NAME}.{EXTENSION}`, where NAME can be anything, and EXTENSION describes the type of data in the file.

We’ve provided you with a few sample patients to get started with, but you should feel free to add data that is useful to your specific Indivo installation.

### 4.10.4 Adding To the Available Sample Data

Adding sample data to Indivo is trivial: simply add files to `settings.SAMPLE_DATA_DIR`, making sure to preserve the directory structure described above. You can either:

- Add data to an existing profile, by dropping new data files into that profile’s directory, or
- Add a new profile, by creating a new subdirectory of `SAMPLE_DATA_DIR`. Make sure to add a demographics document for the new profile.

### 4.11 Indivo Schemas

#### 4.11.1 Introduction

Schemas in Indivo are used to describe valid formats in which data may enter Indivo. Right now, we use XSDs as schemas, since we accept input data only in XML form, but in the future we might extend this to include schemas for validating other formats of data (OWL for RDF, etc.). Note that schemas describe the format of input data only! For information on how data is processed and stored in Indivo, see Data Pipeline.

There are a number of XML standards for medical activities, ranging from the CCR summary to the highly detailed CCD. None of these are particularly well tuned to the needs of a PCHR, where an individual datum may come from a hospital data feed, or from patient-based data entry. The Indivo schemas are built to serve the specific PCHR needs. Importantly, the Indivo schemas use standard coding systems wherever possible. The schemas are also ready for new coding systems as they emerge, especially in the realm of personally-controlled medicine with simplified terminology.

#### 4.11.2 What if I want to store data that doesn’t match an Indivo schema?

Indivo X is designed to accept documents that conform to any XML schema, such as CCR, and even documents that are not XML, i.e. PDFs, MPEG, etc....

XML documents that conform to the built-in schemas can be immediately transformed, via the Indivo X Data Pipeline, into individual datapoints, which can then be queried using the Indivo Reporting API. XML documents that conform to custom schemas are not processed, and therefore cannot be retrieved using the reporting API (though you can still access them with API calls for retrieving unprocessed documents, which will return them in their original XML form.)
If you want to extend Indivo to enable querying over data input according to a new schema, see *Adding Custom Schemas to Indivo*.

### 4.11.3 Namespace and XML Types

All of the default Indivo X document schemas are in a single namespace:

http://indivo.org/vocab/xml/documents#

The use of the trailing # enables simple RDF-like concatenation of namespace prefix and suffix to generate a single type URL. For example, an SDMX document in the Indivo documents namespace will have as its type:

http://indivo.org/vocab/xml/documents#Models

### 4.11.4 Design Rationale for Inclusion vs. Relation

Indivo X brings the ability to relate documents to one another using metadata, rather than document payload. This is particularly important when the payload might not be under the user’s control, i.e. a CCR document. It can also be useful even in the design of new Indivo schemas.

One could imagine separation the prescription information from the medication information, having two documents related to one another rather than one bigger document. However, our design rationale for now is to keep medication and its prescription data in the same XML document because those two chunks of data are generated in the same event. If, at some point, Indivo stores prescription filling information, then it is likely that this information would be more appropriately stored in a separate, linked document.

### 4.11.5 Core Schemas

All schema files and sample instance documents are available at http://indivo.org/vocab/xml/. Note that these schemas are only the ones that come with Indivo by default. Each instance of Indivo might define additional, custom schemas that are not documented here. See *Adding Custom Schemas to Indivo* for instructions on how to add custom schemas to Indivo.

**Metadata and Indivo Internal Data Structures**

**Indivo Document Metadata Schema**

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- didn’t place this in the Indivo namespace because it’s not medical payload -->
<x:schema xmlns:x="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="Principal">
        <xs:sequence>
            <xs:element name="fullname" type="xs:string" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
        <!-- e.g. "fhh@apps.indivo.org" -->
        <xs:attribute name="id" type="xs:string" use="required" />
        <!-- e.g. "userapp" or "account" or "adminapp" -->
        <xs:attribute name="type" type="xs:string" use="required" />
    </xs:complexType>
</xs:schema>
```

4.11. Indivo Schemas
</xs:complexType>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
</xs:complexType>
Example:

<?xml version="1.0" encoding="utf-8" ?>
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d544649703db3a9bc231550a0b0e5bcb1c" size="77">
  <createdAt>2009-05-04T17:05:33</createdAt>
  <creator id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </creator>
  <suppressedAt>2009-05-06T17:05:33</suppressedAt>
  <suppressor id="steve@indivo.org" type="account">
    <fullname>Steve Zabak</fullname>
  </suppressor>
  <original id="14c81023-c84f-496d-8b8e-9438280441d3" />
  <latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
    <label>HBA1C reading</label>
    <status>active</status>
    <neverShare>false</neverShare>
    <relatesTo>
      <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
      <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
    </relatesTo>
    <isRelatedFrom>
      <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
    </isRelatedFrom>
  </latest>
</Document>

**Indivo Account Schema**

An Indivo Account represents a single user of the system, with their basic info and the ways in which they authenticate. It is separate from a record.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <element name="Account">
    <complexType>
      <sequence>
        <element name="secret" type="string" minOccurs="0" maxOccurs="1" />
        <element name="fullName" type="string" minOccurs="1" maxOccurs="1" />
        <element name="contactEmail" type="string" minOccurs="1" maxOccurs="1" />
        <element name="lastLoginAt" type="dateTime" minOccurs="1" maxOccurs="1" />
        <element name="totalLoginCount" type="integer" minOccurs="1" maxOccurs="1" />
        <element name="failedLoginCount" type="integer" minOccurs="1" maxOccurs="1" />
      </sequence>
    </complexType>
  </element>
</schema>
```
Example:

<Account id="joeuser@indivo.example.org">
  <fullName>Joe User</fullName>
  <contactEmail>joeuser@gmail.com</contactEmail>
  <lastLoginAt>2010-05-04T15:34:23Z</lastLoginAt>
  <totalLoginCount>43</totalLoginCount>
  <failedLoginCount>0</failedLoginCount>
  <state>active</state>
  <lastStateChange>2009-04-03T13:12:12Z</lastStateChange>
  <authSystem name="password" username="joeuser"/>
  <authSystem name="hospital_sso" username="Joe_User"/>
</Account>

Indivo PHA Schema

Information describing a Personal Health App (User App). Can be wrapped into a set of Apps.

Schema:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:complexType name="App">
    <xs:sequence>
      <xs:element name="startURLTemplate" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="description" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="autonomous" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      <xs:element name="autonomousReason" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="frameable" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      <xs:element name="ui" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:element name="App" type="App"/>
  <xs:element name="Apps">
    <xs:complexType>
      <xs:sequence>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
Example:

```xml
<App id="problems@apps.indivo.org">
  <startURLTemplate>http://problems.indivo.org/auth/start?record_id={record_id}&amp;carenet_id={carenet_id}</startURLTemplate>
  <name>Problem List</name>
  <description>Managing your problem list</description>
  <autonomous>false</autonomous>
  <frameable>true</frameable>
  <ui>true</ui>
</App>
```

Example of multiple apps:

```xml
<Apps>
  <App id="problems@apps.indivo.org">
    <startURLTemplate>http://problems.indivo.org/auth/start?record_id={record_id}&amp;carenet_id={carenet_id}</startURLTemplate>
    <name>Problem List</name>
    <description>Managing your problem list</description>
    <autonomous>false</autonomous>
    <frameable>true</frameable>
    <ui>true</ui>
  </App>
</Apps>
```

**Indivo Audit Log Schema**

As of Beta 3, the logs will be returned as Indivo Reports according to the *Indivo Reporting Schema*. Each report item will be of type `<AuditEntry>`, as defined below:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="AuditEntry">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="BasicInfo" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="datetime" type="xs:dateTime" use="required" />
            <xs:attribute name="view_func" type="xs:string" use="required" />
            <xs:attribute name="request_successful" type="xs:boolean" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="PrincipalInfo" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="effective_principal" type="xs:string" use="required" />
            <xs:attribute name="proxied_principal" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="Resources" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="carenet_id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<AuditEntry>
  <BasicInfo datetime="2011-04-27T17:32:23Z" view_func="get_document" request_successful="true" />
  <PrincipalInfo effective_principal="myapp@apps.indivohealth.org" proxied_principal="me@indivohealth.org" />
  <Resources carenet_id="" record_id="123" pha_id="" document_id="acd" external_id="" message_id="" />
  <RequestInfo req_url="/records/123/documents/acd/" req_ip_address="127.0.0.1" req_domain="localhost" req_method="GET" />
  <ResponseInfo resp_code="200" />
</AuditEntry>
```

Indivo Carenet Schema

A list of carenets is returned when a user/app wants to know how a document is shared. However, this same list of carenets might be used in a different setting. Thus, the “mode” attribute is optional. It indicates whether sharing in this carenet was done explicitly, or via some implicit auto-share rule.

Schema:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="Carenets">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Carenet" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required" />
            <xs:attribute name="name" type="xs:string" use="required" />
            <xs:attribute name="mode" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Example:

```xml
<Carenets record_id="123">
  <Carenet id="789" name="Work/School" mode="explicit" />
</Carenets>
```

**Indivo Document Status History Schema**

When a document’s status changes (archived, etc..), its history of changes is documented and available in this schema.

**Schema:**

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="DocumentStatusHistory">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DocumentStatus" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="reason" type="xs:string" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
            <xs:attribute name="by" type="xs:string" use="required" />
            <xs:attribute name="at" type="xs:dateTime" use="required" />
            <xs:attribute name="status" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="document_id" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```xml
<DocumentStatusHistory document_id="456">
  <DocumentStatus by="joeuser@indivo.example.org" at="2010-09-03T12:45:12Z" status="archived">
    <reason>no longer relevant</reason>
  </DocumentStatus>
</DocumentStatusHistory>
```

**Indivo Inbox Message Schema**

Indivo messages, sent to accounts (sometimes via a record), are represented with the following schema.

**Schema:**

```xml
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:complexType name="Message">
    <xs:sequence>
      <xs:element name="sender" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="received_at" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
      <xs:element name="read_at" type="xs:dateTime" minOccurs="0" maxOccurs="1" />
      <xs:element name="subject" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="severity" type="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="record" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
      <xs:element name="attachment" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="num" type="xs:integer" use="required" />
          <xs:attribute name="type" type="xs:string" use="required" />
          <xs:attribute name="size" type="xs:integer" use="required" />
          <xs:attribute name="doc_id" type="xs:string" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>

Example:

<Message id="879">
  <sender>doctor@example.indivo.org</sender>
  <received_at>2010-09-04T14:12:12Z</received_at>
  <read_at>2010-09-04T17:24Z</read_at>
  <subject>your test results are looking good</subject>
  <severity>normal</severity>
  <record id="123" />
  <attachment num="1" type="http://indivo.org/vocab/xml/documents#Lab" size="12546" />
</Message>
</Messages>

Another Example:

<Message id="879">
  <sender>doctor@example.indivo.org</sender>
  <received_at>2010-09-04T14:12:12Z</received_at>
</Message>
<subject>your test results are looking good</subject>
<severity>normal</severity>
<record id="123" />
<attachment num="1" type="http://indivo.org/vocab/xml/documents#Lab" size="12546" />
</Message>

Indivo Notification Schema

The Indivo Healthfeed includes notifications, represented by this schema:

Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="Notifications">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Notification">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="sender" type="xs:string" minOccurs="1" maxOccurs="1" />
              <xs:element name="received_at" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
              <xs:element name="content" type="xs:string" minOccurs="1" maxOccurs="1" />
              <xs:element name="record" minOccurs="1" maxOccurs="1">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:string" use="required" />  
                  <xs:attribute name="label" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
              <xs:element name="document" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:string" use="required" />  
                  <xs:attribute name="label" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```
<Notifications>
  <Notification id="468">
    <sender>labs@apps.indivo.org</sender>
    <received_at>2010-09-03T15:12:12Z</received_at>
    <content>A new lab result has been delivered to your account</content>
    <record id="123" label="Joe User" />
    <document id="579" label="Lab Test 2" />
  </Notification>
</Notifications>
```
Indivo Permissions Schema
Coming Soon...

Indivo Record Schema

The basic info for an Indivo record. Some of the attributes are there for indicating sharing relationships.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="Record"><xs:complexType><xs:sequence>
    <xs:element name="demographics" minOccurs="1" maxOccurs="1">
      <xs:complexType>
        <xs:attribute name="document_id" type="xs:string" use="required" />
      </xs:complexType>
    </xs:element>
    <xs:element name="created" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:attribute name="at" type="xs:dateTime" use="optional" />
        <xs:attribute name="by" type="xs:string" use="optional" />
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="label" type="xs:string" use="required" />
  <xs:attribute name="shared" type="xs:boolean" use="optional" />
  <xs:attribute name="carenet_id" type="xs:string" use="optional" />
  <xs:attribute name="carenet_name" type="xs:string" use="optional" />
  <xs:attribute name="role_label" type="xs:string" use="optional" />
</xs:complexType>
</xs:element>
</xs:schema>
```

Example:

```xml
<Record id="123" label="Joe User" shared="true">
  <demographics document_id="467" />
  <created at="2010-10-23T10:23:34Z" by="indivoconnector@apps.indivo.org" />
</Record>
```

Indivo Request Token Schema

The Indivo UI Server needs to manage request tokens for apps so that it can display the appropriate authorization screens. This schema makes use of the Indivo PHA Schema.
Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:include schemaLocation="../pha/pha.xsd" />
  <xs:element name="RequestToken">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="record" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
        <xs:element name="carenet" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
        <xs:element name="kind" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="App" type="App" minOccurs="1" maxOccurs="1" />
      </xs:sequence>
      <xs:attribute name="token" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```xml
<RequestToken token="XYZ">
  <record id="123" />
  <carenet />
  <kind>new</kind>
  <App id="problems@apps.indivo.org">
    <name>Problem List</name>
    <description>Managing your list of problems</description>
    <autonomous>false</autonomous>
    <frameable>true</frameable>
    <ui>true</ui>
  </App>
</RequestToken>
```

**Indivo Coded Values**

A coded value is a value taken from a coding system. It consists of a reference to the coding system (a URL), the code value, and the human-readable string. When the coding system is not used but a manual value is entered, the coding system and coded value are absent, leaving only the human-readable string.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

4.11. Indivo Schemas
• When a document comes into Indivo, its coded values may be expanded (with abbreviation and element content) or not (just the code and coding system).

• We will encourage applications to provide expanded coded values, but this will not be required.

• Reports will provide abbreviations and full names for all relevant codes by looking up against Indivo-stored copies of the coding systems. Documents will not be modified from what the sources send us, to follow the principles of store exactly the original data source (that’s required because the documents might be digitally signed.)

• Reports can flag codes whose abbreviations and full names do not match the coding system data (but we always show by default what the document says, we trust the source, not the coding system.)

• We then need a code lookup API for viewing single documents.

• codes.indivo.org will provide an API for interpreting codes.

Indivo Data Values Schema

Schema:

<?xml version="1.0" encoding="ISO-8859-1" ?>

<x:schema name="Range">

<x:schema name="ValueAndUnit">

<!-- an ordinal, i.e. "2+" is coded using textValue -->
<x:schema name="ValueAndUnit">


<!-- a result is an abstract base, only subtypes can be used -->
<xs:complexType name="Result" abstract="true">
  <xs:sequence>
    <!-- HL7 defines flag types -->
    <xs:element name="flag" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ResultInRange">
  <xs:complexContent>
    <xs:extension base="indivo:Result">
      <xs:sequence>
        <xs:element name="valueAndUnit" type="indivo:ValueAndUnit" minOccurs="1" maxOccurs="1" />
        <xs:element name="normalRange" type="indivo:Range" minOccurs="0" maxOccurs="1" />
        <!-- nontoxicrange as in, if it's outside the range, it's toxic -->
        <xs:element name="nonCriticalRange" type="indivo:Range" minOccurs="0" maxOccurs="1" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- untyped strings, e.g. "positive" -->
<xs:complexType name="ResultInSet">
  <xs:complexContent>
    <xs:extension base="indivo:Result">
      <xs:sequence>
        <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1" />
        <!-- the options should be listed in sensible order -->
        <xs:element name="option" minOccurs="0" maxOccurs="unbounded">
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="normal" type="xs:boolean" use="required" />
              <xs:attribute name="description" type="xs:string" use="optional" />
            </xs:extension>
          </xs:simpleContent>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Concentration">
  <xs:complexContent>
    <xs:extension base="indivo:ValueOrRange" />
  </xs:complexContent>
</xs:complexType>

4.11. Indivo Schemas
Indivo Provider Schema

A provider is typically an MD with an institution affiliation.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
    xmlns:indivo="http://indivo.org/vocab/xml/documents#"
    elementFormDefault="qualified">
  <complexType name="Provider">
    <sequence>
      <element name="name" type="string" minOccurs="1" maxOccurs="1" />
      <element name="institution" type="string" minOccurs="1" maxOccurs="1" />
    </sequence>
  </complexType>
  <complexType name="LabProvider">
    <sequence>
      <element name="name" type="string" minOccurs="1" maxOccurs="1" />
      <element name="address" type="string" minOccurs="1" maxOccurs="1" />
    </sequence>
  </complexType>
  <complexType name="Signature">
    <sequence>
      <element name="at" type="dateTime" minOccurs="0" maxOccurs="1" />
      <element name="provider" type="indivo:Provider" minOccurs="1" maxOccurs="1" />
    </sequence>
  </complexType>
</schema>
```

Reporting

Indivo Reporting Schema

See the schema for Indivo Document Metadata Schema and specific indivo:doc schemas.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
    elementFormDefault="qualified">
  <xs:complexType name="Report">
    <xs:sequence>
      <xs:element name="Meta" minOccurs="1" maxOccurs="1"/>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Document" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
      </xs:complexType>
    </xs:sequence>
  </complexType>
</xs:schema>
```
<xs:element name="Item" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="1" maxOccurs="1">
        <xs:element name="Allergy" />
        <xs:element name="Equipment" />
        <xs:element name="Immunization" />
        <xs:element name="LabReport" />
        <xs:element name="Medication" />
        <xs:element name="Problem" />
        <xs:element name="Procedure" />
        <xs:element name="SimpleClinicalNote" />
        <xs:element name="VitalSign" />
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Reports">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Summary" minOccurs="1" maxOccurs="1">
        <xs:complexType>
          <xs:attribute name="total_document_count" use="optional" type="xs:int" />
          <xs:attribute name="limit" use="optional" type="xs:int" />
          <xs:attribute name="offset" use="optional" type="xs:int" />
          <xs:attribute name="order_by" use="optional" type="xs:string" />
        </xs:complexType>
      </xs:element>
      <xs:element name="QueryParams" minOccurs="1" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="GroupBy" minOccurs="0" maxOccurs="1">
              <xs:complexType>
                <xs:attribute name="value" use="required" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <xs:element name="DateGroup" minOccurs="0" maxOccurs="1">
              <xs:complexType>
                <xs:attribute name="value" use="required" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <xs:element name="AggregateBy" minOccurs="0" maxOccurs="1">
              <xs:complexType>
                <xs:attribute name="value" use="required" type="xs:string" />
              </xs:complexType>
            </xs:element>
            <xs:element name="DateRange" minOccurs="0" maxOccurs="1">
              <xs:complexType>
                <xs:attribute name="value" use="required" type="xs:string" />
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

4.11. Indivo Schemas
Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*" />
    <Filters>
      <Filter name="lab_type" value="hematology"/>
    </Filters>
  </QueryParams>
  <Report>
    <Meta>
      <Document id="261ca370-927f-41af-b001-7b615c7a468e" type="http://indivo.org/vocab/xml/documents#Lab">
        <createdAt>2011-05-02T17:48:13Z</createdAt>
        <creator id="mymail@mail.ma" type="Account">
          <fullname>full name</fullname>
        </creator>
        <original id="261ca370-927f-41af-b001-7b615c7a468e"/>
        <label>testing</label>
        <status>active</status>
        <nevershare>false</nevershare>
      </Document>
    </Meta>
    <Item>
      <LabReport xmlns="http://indivo.org/vocab/xml/documents#">
        <dateMeasured>1998-07-16T12:00:00Z</dateMeasured>
        <labType>hematology</labType>
        <laboratory>
          <name>Quest</name>
          <address>300 Longwood Ave, Boston MA 02215</address>
        </laboratory>
        <comments>was looking pretty sick</comments>
        <firstPanelName>CBC</firstPanelName>
      </LabReport>
    </Item>
  </Report>
</Reports>
```
Indivo Aggregate Report Schema

This schema describes report items returned in aggregate form.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
  elementFormDefault="qualified">

  <xs:complexType name="AggregateReport">
    <xs:attribute name="value" type="xs:string" use="required"/>
    <xs:attribute name="group" type="xs:string" use="optional"/>  
  </xs:complexType>

  <xs:element name="AggregateReport" type="indivo:AggregateReport"/>
</xs:schema>
```

Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
```

Indivo Generic Aggregate Reports Schema

This schema describes report items returned in XML aggregate form from Generic Reports.

Schema:

```xml
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:complexType name="AggregateReport">
    <xs:attribute name="value" type="xs:string" use="required" />
    <xs:attribute name="group" type="xs:string" use="optional" />
</xs:complexType>

<xs:element name="AggregateReports">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="AggregateReport" type="AggregateReport" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>

Example:

<?xml version="1.0" encoding="utf-8" ?>
<AggregateReports>
    <AggregateReport value="1" group="2009-07" />
    <AggregateReport value="4" group="2009-08" />
</AggregateReports>

Special Documents

Indivo Document Demographics Schema

Schema:

<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:indivo="http://indivo.org/vocab/xml/documents#"
    elementFormDefault="qualified"
    targetNamespace="http://indivo.org/vocab/xml/documents#">

<xs:complexType name="Name">
    <xs:sequence>
        <xs:element name="familyName" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="givenName" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="middleName" type="xs:string" minOccurs="0" maxOccurs="1" />
        <xs:element name="prefix" type="xs:string" minOccurs="0" maxOccurs="1" />
        <xs:element name="suffix" type="xs:string" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="PhoneType">
    <xs:restriction base="xs:normalizedString">
        <xs:enumeration value="h"/>
        <xs:enumeration value="w"/>
        <xs:enumeration value="c"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="GenderType">
<xs:restriction base="xs:normalizedString">
  <xs:enumeration value="female"/>
  <xs:enumeration value="male"/>
</xs:restriction>
</xs:simpleType>

<xs:complexType name="Telephone">
  <xs:sequence>
    <xs:element name="type" type="indivo:PhoneType" minOccurs="0" maxOccurs="1" />
    <xs:element name="number" type="xs:string" minOccurs="1" maxOccurs="1" />
    <xs:element name="preferred" type="xs:boolean" minOccurs="0" maxOccurs="1" default="false"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Address">
  <xs:sequence>
    <xs:element name="country" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="city" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="postalCode" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="region" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="street" type="xs:string" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:element name="Demographics">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="dateOfBirth" type="xs:date" minOccurs="1" />
      <xs:element name="gender" type="indivo:GenderType" minOccurs="1" />
      <xs:element name="email" type="xs:string" minOccurs="0" />
      <xs:element name="ethnicity" type="xs:string" minOccurs="0" />
      <xs:element name="preferredLanguage" type="xs:string" minOccurs="0" />
      <xs:element name="race" type="xs:string" minOccurs="0" />
      <xs:element name="Name" type="indivo:Name" minOccurs="1"/>
      <xs:element name="Telephone" type="indivo:Telephone" minOccurs="0" maxOccurs="2" />
      <xs:element name="Address" type="indivo:Address" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Demographics xmlns="http://indivo.org/vocab/xml/documents#">
  <dateOfBirth>1939-11-15</dateOfBirth>
  <gender>male</gender>
  <email>test@fake.org</email>
  <ethnicity>Scottish</ethnicity>
  <preferredLanguage>EN</preferredLanguage>
  <race>caucasian</race>
  <Name>
    <familyName>Wayne</familyName>
    <givenName>Bruce</givenName>
    <middleName>Quentin</middleName>
    <prefix>Mr</prefix>
    <suffix>Jr</suffix>
  </Name>
  <Telephone>
```
<type>h</type>
<number>555-5555</number>
<preferred>true</preferred>
</Telephone>
<Telephone>
<type>c</type>
<number>555-6666</number>
</Telephone>
<Address>
<country>USA</country>
<city>Gotham</city>
<postalCode>90210</postalCode>
<region>secret</region>
<street>1007 Mountain Drive</street>
</Address>
</Demographics>

**Indivo Document Annotation Schema**

Medical documents can be annotated in Indivo, with a document that’s added in relation to the annotated document. This is a simple schema for these text-based annotations.

Schema:
Coming Soon!

Example:
Coming Soon!

The relationship to the annotated document is maintained in Indivo metadata, and the annotation can, optionally, store the SHA256 hash of the referenced document for robustness.

**Medical Documents**

**Indivo Document Schema: Procedure**

A procedure is effectively a surgical event (though some are not exactly surgical.) See also the schema for *Indivo Coded Values* and for *Indivo Provider Schema*.

Schema:
```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
 xmlns:indivo="http://indivo.org/vocab/xml/documents#" elementFormDefault="qualified">
  <include schemaLocation="../../common/codes.xsd" />
  <include schemaLocation="../../common/provider.xsd" />
  <element name="Procedure">
    <complexType>
      <sequence>
        <element name="datePerformed" type="dateTime" minOccurs="0" maxOccurs="1" />
        <element name="name" type="indivo:CodedValue" minOccurs="1" maxOccurs="1" />
        <element name="provider" type="indivo:Provider" minOccurs="0" />
        <element name="location" type="string" minOccurs="0" maxOccurs="1" />
      </sequence>
    </complexType>
  </element>
</schema>
```
<element name="comments" type="string" minOccurs="0" maxOccurs="1" />
</sequence>
</complexType>
</element>
</schema>

Example:

<Procedure xmlns="http://indivo.org/vocab/xml/documents#">
 <datePerformed>2009-05-16T12:00:00</datePerformed>
 <name type="http://codes.indivo.org/procedures#" value="85" abbrev="append">Appendectomy</name>
 <provider>
  <name>Kenneth Mandl</name>
  <institution>Children's Hospital Boston</institution>
 </provider>
</Procedure>

Indivo Document Schema: Equipment

Schema:

<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://indivo.org/vocab/xml/documents#"
  xmlns:indivo="http://indivo.org/vocab/xml/documents#"
  elementFormDefault="qualified">
  <element name="Equipment">
    <complexType>
      <sequence>
        <element name="dateStarted" type="date" minOccurs="0" maxOccurs="1" />
        <element name="dateStopped" type="date" minOccurs="0" maxOccurs="1" />
        <element name="type" type="string" minOccurs="0" maxOccurs="1" />
        <element name="name" type="string" minOccurs="1" maxOccurs="1" />
        <element name="vendor" type="string" minOccurs="0" maxOccurs="1" />
        <element name="id" type="string" minOccurs="0" maxOccurs="1" />
        <element name="description" type="string" minOccurs="0" maxOccurs="1" />
        <element name="specification" type="string" minOccurs="0" maxOccurs="1" />
        <element name="certification" type="string" minOccurs="0" maxOccurs="1" />
      </sequence>
    </complexType>
  </element>
</schema>

Example:

<Equipment xmlns="http://indivo.org/vocab/xml/documents#">
 <dateStarted>2009-02-05</dateStarted>
 <dateStopped>2010-06-12</dateStopped>
 <type>cardiac</type>
 <name>Pacemaker</name>
 <vendor>Acme Medical Devices</vendor>
 <id>167-ABC-23</id>
 <description>it works</description>
 <specification>blah blah blah</specification>
</Equipment>
A full clinical note needs to contain a number of coded problems, etc. Some hospital systems do not have fully normalized clinical notes, in which case they can use this schema to store some simple attributes and the main free-form text of the note.

See also Indivo Coded Values and Indivo Provider Schema.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
   elementFormDefault="qualified">
   <include schemaLocation="../../common/codes.xsd" />
   <include schemaLocation="../../common/provider.xsd" />
   <!-- this is mostly a chunk o' text. More normalized clinical notes will be in a diff schema -->
   <element name="SimpleClinicalNote">
      <complexType>
         <sequence>
            <element name="dateOfVisit" type="dateTime" minOccurs="1" maxOccurs="1" />
            <element name="finalizedAt" type="dateTime" minOccurs="0" maxOccurs="1" />
            <element name="visitType" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
            <element name="visitLocation" type="string" minOccurs="0" maxOccurs="1" />
            <element name="specialty" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
            <element name="signature" type="indivo:Signature" minOccurs="1" maxOccurs="unbounded" />
            <element name="chiefComplaint" type="string" minOccurs="0" maxOccurs="1" />
            <element name="content" minOccurs="0" maxOccurs="1" type="string" />
         </sequence>
      </complexType>
   </element>
</schema>
```

Example:

```xml
<SimpleClinicalNote xmlns="http://indivo.org/vocab/xml/documents#">
   <dateOfVisit>2010-02-02T12:00:00Z</dateOfVisit>
   <finalizedAt>2010-02-03T13:12:00Z</finalizedAt>
   <visitType type="http://codes.indivo.org/visit-types#" value="acute">Acute Care</visitType>
   <visitLocation>Longfellow Medical</visitLocation>
   <specialty type="http://codes.indivo.org/specialties#" value="hem-onc">Hematology/Oncology</specialty>
   <signature>
      <at>2010-02-03T13:12:00Z</at>
      <provider>
         <name>Kenneth Mandl</name>
         <institution>Children’s Hospital Boston</institution>
      </provider>
   </signature>
   <signature>
      <provider>
         <name>Isaac Kohane</name>
         <institution>Children’s Hospital Boston</institution>
      </provider>
   </signature>
</SimpleClinicalNote>
```
<chiefComplaint>stomach ache</chiefComplaint>
<content>
    Patient presents with ...
</content>
</SimpleClinicalNote>

**Indivo Document Schema: SDMX**

For any *data model* in Indivo that can be represented in *SDML*, we will accept data in the form of *SDMX*.

Schema:

```xml
    elementFormDefault="qualified"
    xmlns:indivo="http://indivo.org/vocab/xml/documents#">

  <complexType name="ModelType">
    <sequence>
      <element name="Field" minOccurs="0" maxOccurs="unbounded">
        <complexType mixed="true">
          <choice>
            <element name="Model" type="indivo:ModelType" minOccurs="0" maxOccurs="1"/>
            <element name="Models" type="indivo:ModelsType" minOccurs="0" maxOccurs="1"/>
          </choice>
          <attribute name="name" type="string" use="required"/>
        </complexType>
      </element>
    </sequence>
    <attribute name="name" type="string" use="required"/>
    <attribute name="documentId" type="string" use="optional"/>
  </complexType>

  <complexType name="ModelsType">
    <sequence>
      <element name="Model" type="indivo:ModelType" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <element name="Models" type="indivo:ModelsType"/>
</schema>
```

Example:

```xml
<Models xmlns="http://indivo.org/vocab/xml/documents#">
  <Model name="TestMedication" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
    <Field name="date_started">2010-10-01T00:00:00Z</Field>
    <Field name="name">ibuprofen</Field>
    <Field name="brand_name">Advil</Field>
    <Field name="date_stopped">2010-10-31T00:00:00Z</Field>
    <Field name="prescription">
      <Model name="TestPrescription" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
        <Field name="prescribed_by_name">Kenneth D. Mandl</Field>
        <Field name="prescribed_by_institution">Children’s Hospital Boston</Field>
        <Field name="prescribed_on">2010-09-30T00:00:00Z</Field>
        <Field name="prescribed_stop_on">2010-10-31T00:00:00Z</Field>
      </Model>
    </Field>
    <Field name="fills">
      <Models/>
    </Field>
  </Model>
</Models>
```
4.11.6 Adding Custom Schemas to Indivo

As of version 1.1 of Indivo X, we’ve added a feature that makes it much easier to add (in a drag-and-drop fashion) new supported schemas to an instance of Indivo. Adding a new schema to Indivo involves:

- Creating the schema
- Mapping the schema to Indivo’s Data Models
- Dropping the schema into the filesystem

Creating the Schema

Indivo currently accepts schemas only in XSD form. There are numerous tutorials and tools on the web to help you create an XSD, so we won’t presume to tell you how you should do it. What matters is that you build an XSD which can validate documents for further processing in the Indivo data pipeline.

Mapping the Schema to Data Models

In order to arrive at processed facts that can be queried and retrieved, you’ll need to a way to transform documents matching your schema into a form understood by Indivo. We call this tool (intuitively) a Transform, and you can learn how to build one here.

If the data in your new schema doesn’t fit into any of the Indivo Data Models, and you want to add a new data-model to Indivo, see Adding Custom Data-Models to Indivo.

Dropping the Schema into the Filesystem

Indivo schemas currently have the following layout on the filesystem:

```
indivo_server/
    indivo/
        ...
        schemas/
            tools/
            metadata/
            data/
                common/
                output/
```

```xml
<Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
    <Field name="date_filled">2010-10-01T00:00:00Z</Field>
    <Field name="supply_days">15</Field>
    <Field name="filled_at_name">CVS</Field>
</Model>
<Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
    <Field name="date_filled">2010-10-16T00:00:00Z</Field>
    <Field name="supply_days">15</Field>
    <Field name="filled_at_name">CVS</Field>
</Model>
```
The `indivo/schemas/data/core/` directory contains all of our built-in schemas, and you shouldn’t modify it. Since you are ‘contributing’ a schema to Indivo, add your schema to the `indivo/schemas/data/contrib/` directory. Simply:

- **Create a new subdirectory under** `indivo/schemas/data/contrib/`.
- **Drop the following files into that directory:**
  - `schema.xsd`: Your schema. This file **MUST BE NAMED SCHEMA.XSD** to be identified as a schema.
  - `transform.[xslt | py]`: Your transform. This file **MUST BE NAMED ‘transform’** to be identified.
  - `sample.xml` *(optional)*: A sample document that should validate against your schema. This is optional, but is a good way to make sure your schema works as intended. If you have one or more sample xml files in your directory (you can name them anything, as long as the filename ends in `.xml`), you can make sure that they all validate by running:
    ```
    cd indivo_server/indivo/schemas
    python utils/validate.py
    ```
    This will validate all sample documents against their schema: you should see ‘ok’ at the end of each line of output if there were no errors.
- **Restart Indivo for your changes to take effect. Your final directory structure should now look something like:**

```
indivo_server/
    indivo/
    ...
    schemas/
    util/
    metadata/
    data/
    common/
    output/
    core/
    sdmx/
        schema.xsd
        transform.[xslt | py]
        sdmx.xml
    ...
    contrib/
    your_schema/
        schema.xsd
        transform.[xslt | py]
        your_example1.xml
        your_example2.xml
    ...
```
4.12 Indivo Transforms

4.12.1 Introduction

For the pipeline to be functional, data must be transformed from its original format into processed medical facts ready to be stored in the database. Each schema in Indivo therefore defines a *transform* that can be applied to any document that validates against the schema.

4.12.2 Transform Outputs

The ultimate output of the transformation step in the data pipeline is a set of *Fact objects* ready for storage in the database. However, technologies like XSLT are incapable of producing python objects as output. We looked around for a simple, standard way of modeling data that would meet our needs, and came up empty (though we’re open to suggestions if you think you have the silver bullet). As a result, we’ve created our own language, *Indivo Simple Data Modeling Language (SDML)*, to both define our data models and represent documents (in XML or JSON) that match them.

Thus, transforms may output data in any of the following formats:

- Simple Data Model JSON (SDMJ)
- Simple Data Model XML (SDMX)
- Python Fact objects

Outputs are validated on a per-datamodel-basis. For data model definitions and example outputs, see *Indivo Data Models*.

4.12.3 Types of Transforms

Indivo currently accepts Transforms in two formats:

- XSLT documents
- Python classes

This may change as Indivo begins to accept data in more, varied formats.

**XSLTs**

We won’t cover XSLTs in any detail here, as their format and use is clearly outlined in the specification. Since XSLT is traditionally used to transform XML to XML, the most natural output format for XSLTs is *SDMX*.

**Python**

For those unskilled in the arts of XSLT, we also allow transforms to be defined using python. To define a transform, simply subclass *indivo.document_processing.BaseTransform* and define a valid transformation method. Valid methods are:

```python
BaseTransform.to_facts(doc_etree)
```

Transform an etree into a list of Indivo Fact objects.

Subclasses should implement this method, which takes an `lxml.etree._ElementTree` (the result of calling `etree.parse()`), and returns a list of `indivo.models.Fact` subclasses.
As an example, here’s how you might implement this for a transform from our Simple Clinical Note Schema to our Simple Clinical Note Data Model:

```python
from indivo.document_processing import BaseTransform
from indivo.models import SimpleClinicalNote
from lxml import etree

NS = "http://indivo.org/vocab/xml/documents#"

class Transform(BaseTransform):
    def to_facts(self, doc_etree):
        args = self._get_data(doc_etree)
        # Create the fact and return it
        # Note: This method must return a list
        return [SimpleClinicalNote(**args)]

    def _tag(self, tagname):
        return "{%s}%s{%s}"%(NS, tagname)

    def _get_data(self, doc_etree):
        """ Parse the etree and return a dict of key-value pairs for object construction. """
        ret = {}
        _t = self._tag

        # Get the date_of_visit
        ret["date_of_visit"] = doc_etree.findtext(_t("dateOfVisit"))

        # Get the date finalized
        ret["finalized_at"] = doc_etree.findtext(_t("finalizedAt"))

        # Get the visit_type
        visit_type_node = doc_etree.find(_t("visitType"))
        ret["visit_type"] = visit_type_node.text
        ret["visit_type_type"] = visit_type_node.get("type")
        ret["visit_type_value"] = visit_type_node.get("value")
        ret["visit_type_abbrev"] = visit_type_node.get("abbrev")

        # Get the visit location
        ret["visit_location"] = doc_etree.findtext(_t("visitLocation"))

        # Get the specialty of the clinician
        specialty_node = doc_etree.find(_t("specialty"))
        ret["specialty"] = specialty_node.text
        ret["specialty_type"] = specialty_node.get("type")
        ret["specialty_value"] = specialty_node.get("value")
        ret["specialty_abbrev"] = specialty_node.get("abbrev")

        # get the signature
        signature_node = doc_etree.find(_t("signature"))
        provider_node = signature_node.find(_t("provider"))
        ret["signed_at"] = signature_node.findtext(_t("at"))
        ret["provider_name"] = provider_node.findtext(_t("name"))
        ret["provider_institution"] = provider_node.findtext(_t("institution"))

        # get the chief complaint
        ret["chief_complaint"] = doc_etree.findtext(_t("chiefComplaint"))
```

4.12. Indivo Transforms
get the content of the note
ret['content'] = doc_etree.findtext(_t('content'))

return ret

BaseTransform.to_sdmj(doc_etree)
Transform an etree into a string of valid Simple Data Model JSON.
Subclasses should implement this method, which takes an lxml.etree._ElementTree (the result of calling etree.parse()), and returns a string in valid SDMJ format.
As an example, here’s how you might implement this for a transform from our Simple Clinical Note Schema to our Simple Clinical Note Data Model. Note the reuse of the _get_data() function from above:

from indivo.document_processing import BaseTransform

SDMJ_TEMPLATE = '''
{
    "__modelname__": "SimpleClinicalNote",
    "date_of_visit": "%(date_of_visit)s",
    "finalized_at": "%(finalized_at)s",
    "visit_type": "%(visit_type)s",
    "visit_type_type": "%(visit_type_type)s",
    "visit_type_value": "%(visit_type_value)s",
    "visit_type_abbrev": "%(visit_type_abbrev)s",
    "visit_location": "%(visit_location)s",
    "specialty": "%(specialty)s",
    "specialty_type": "%(specialty_type)s",
    "specialty_value": "%(specialty_value)s",
    "specialty_abbrev": "%(specialty_abbrev)s",
    "signed_at": "%(signed_at)s",
    "provider_name": "%(provider_name)s",
    "provider_institution": "%(provider_institution)s",
    "chief_complaint": "%(chief_complaint)s",
    "content": "%(content)s"
}
'''

class Transform(BaseTransform):
    def to_sdmj(self, doc_etree):
        args = self._get_data(doc_etree)
        return SDMJ_TEMPLATE%args

BaseTransform.to_sdmx(doc_etree)
Transform an etree into a string of valid Simple Data Model XML.
Subclasses should implement this method, which takes an lxml.etree._ElementTree (the result of calling etree.parse()), and returns another lxml.etree._ElementTree instance representing an XML document in valid SDMX format.
As an example, here’s how you might implement this for a transform from our Simple Clinical Note Schema to our Simple Clinical Note Data Model. Note the reuse of the _get_data() function from above:

from indivo.document_processing import BaseTransform
from lxml import etree
from StringIO import StringIO

SDMX_TEMPLATE = '''
<Models>

</Models>

<Models>
<Model name="SimpleClinicalNote">
  <Field name="date_of_visit">%(date_of_visit)s</Field>
  <Field name="finalized_at">%(finalized_at)s</Field>
  <Field name="visit_type">%(visit_type)s</Field>
  <Field name="visit_type_type">%(visit_type_type)s</Field>
  <Field name="visit_type_value">%(visit_type_value)s</Field>
  <Field name="visit_type_abbrev">%(visit_type_abbrev)s</Field>
  <Field name="visit_location">%(visit_location)s</Field>
  <Field name="specialty">%(specialty)s</Field>
  <Field name="specialty_type">%(specialty_type)s</Field>
  <Field name="specialty_value">%(specialty_value)s</Field>
  <Field name="specialty_abbrev">%(specialty_abbrev)s</Field>
  <Field name="signed_at">%(signed_at)s</Field>
  <Field name="provider_name">%(provider_name)s</Field>
  <Field name="provider_institution">%(provider_institution)s</Field>
  <Field name="chief_complaint">%(chief_complaint)s</Field>
  <Field name="content">%(content)s</Field>
</Model>

class Transform(BaseTransform):
    def to_sdmx(self, doc_etree):
        args = self._get_data(doc_etree)
        return etree.parse(StringIO(SDMX_TEMPLATE%args))

4.12.4 Adding Custom Transforms to Indivo

Associating a new transform with an Indivo-supported schema is simple:

- Write your transform, as either an XSLT or a Python module, as described above.
- Drop the file containing your transform (transform.xslt or transform.py: make sure to name the file ‘transform’) into the directory containing the schema. See Adding Custom Schemas to Indivo for more details.
- Make sure to restart Indivo after moving transform files around, or the changes won’t take effect.
5.1 Architecture Overview

This document provides a basic overview of the Indivo X system architecture. This document should be read before continuing on to *Indivo Authentication* and *the Indivo API*.

5.1.1 Basic Indivo Concepts

*Indivo Record*: the complete set of medical information stored by Indivo about a single individual.

*Indivo Account*: a username/password to log into Indivo. One account may be able to access any number of Indivo Records, and one Indivo Record may be accessible by multiple Indivo Accounts.

*Indivo Document*: a piece of medical information stored in an Indivo Record.

5.1.2 Components

Indivo X comprises multiple components, each running as its own web server. Small installations may choose to install multiple components on a single physical server. The Indivo X Server is the core of the system; other components, including the Indivo User Interface, can be easily substituted by custom implementations.
Indivo X Documentation, Release 2.0.0

Indivo X Server

For a given Indivo installation, the Indivo X server:

- stores all Indivo account information, as well as the medical records and documents,
- is responsible for authentication and authorization before granting access to Indivo data,
- exposes an API for access by administrative and user applications, and by the Indivo User Interface.

Indivo User Interface / Indivo Chrome

The Indivo User Interface, also known as the “Indivo Chrome”, implements the web-based visual interface that an Indivo user will view and use. The branding/colors/details of the user interface are all controlled by the Indivo Chrome. Indivo Chrome connects to Indivo X using the standard Indivo API, including some specific calls accessible only to the Chrome component.

Indivo X will ship with a default implementation of the Indivo Chrome which can be customized while maintaining a clean interface to the Indivo X API. Customizations are encouraged for re-branding or for entirely different devices, e.g. iPhone.

(The term “Chrome” is often used to describe the visual portions of a web browser that are part of the web browser itself, and not part of the web site content, e.g. the back button. Here, with “Indivo Chrome”, we mean the Indivo user interface that is part of the core Indivo service, not part of a user application that extends Indivo functionality.)
Administrative Application

An Admin Application can connect to Indivo X and

- create new Indivo accounts and records
- reset of passwords
- manage ownership of records, i.e. assigning an account as the owner of a record.

An admin application cannot access medical data, it can only manage a record’s metadata. An admin app is thus ideal for a hospital administrator, an Indivo help-desk staffer, a research administrator, etc.

User Application / Personal Health Application

A user application, or Personal Health Application, is an application that Indivo users manually add to their record to provide incremental functionality. Examples of PHA functionality include:

- Diabetes management
- Genomic data display
- Clinical trial matching and messaging

User applications generally provide a web interface to the Indivo user, while connecting to the user’s Indivo record directly with the Indivo X Server. Users are fully in control of what data a user application can access. They can, at any time, change those permissions or remove the application entirely. Thus, an Indivo user application connects to Indivo in much the same way that a Facebook application connects to Facebook.

5.1.3 Communication Protocols

All communication between components is over HTTPS, with an API that abides by the REST design philosophy. Authentication is via OAuth.

More about Indivo Authentication.

More about the Indivo API.

5.2 Access Control

After the credentials of a request have been verified using OAuth, incoming requests must only be allowed to access and manipulate data in ways consistent with their credentials. The Indivo access control module handles this authorization.

5.2.1 Key Goals

The aim is to accomplish the following:

- Access Control abstracted from business logic.
- Deny-by-default Access Control: if an Indivo view is not explicitly given permissions, it cannot be executed by anyone.
- View-function-level access control: Indivo buys fully into the REST model: namely, if 2 users can access the same view function, they will get the same response from it. This makes the view function a natural level of granularity for access control.
5.2.2 Example Usage

In order to illustrate the desired behavior of the system, we give an example of what a developer must do upon implementing a new view function. For this example, the view function will be ‘get_carenet_document’, which takes a carenet and a document id and returns the document. The developer must:

- Identify the set of role predicates that define access to the view function. In this case, anyone in the same carenet as the document should be able to access the document, so the role predicate IsInCarenet defines access to the view function.

- Build an access function, which takes the view function arguments and the principal and returns a boolean indicating whether the principal may access the view function. For example:

  ```python
def carenet_document_access(principal, carenet):
    return principal.isInCarenet(carenet)
  ```

- Bind the view function to the access function by creating a new AccessRule object:

  ```python
  AccessRule('Carenet Document Access', carenet_document_access, [get_carenet_document])
  ```

- Note that the third argument to the AccessRule constructor is a list of view functions. This means that if there are other view functions which allow the same access as ‘get_carenet_document’, we could group them in the AccessRule. For example:

  ```python
  AccessRule('Carenet Document Access', carenet_document_access,
             [get_carenet_document, get_carenet_immunization_list, get_carenet_medication_list])
  ```

And that’s it! Now any principle in a carenet may attempt to get documents in the carenet. Note that the call may still return a 404 if the desired document isn’t actually in the referenced carenet—but this not an access control issue. Also note that if the developer forgets to define a mapping from a view function to an access rule, all attempts to access the view will result in a 403 Access Denied error.

5.2.3 Components of the system

As illustrated above, there are several components to the new access control system:

- An interface of role predicates that must be implemented by each subclass of Principal (right now MachineApp, AccessToken, PHA, Account, ReqToken, NoUser). These roles encapsulate the relationships between a principle and data, and include (but are not limited to at this point in time):
  - isProxiedByApp
  - createdAccount
  - createdRecord
  - ownsRecord
  - scopedToRecord
  - fullySharesRecord
  - isInCarenet

A role predicate is defined as a function that returns true if the principal possesses that role with respect to the passed data, and false if it doesn’t. Each principal must implement all role predicates that apply to it. The
abstract Principle class will provide a default implementation that always returns False, so unimplemented roles will deny by default.

- A set of access functions, which evaluate one or more role predicates above to determine whether a principal may access data.
- A set of AccessRule objects, which preserve mappings from groups of view functions to access functions.

### 5.2.4 Authorization Pipeline

When an incoming request is processed by the access control system, the following steps occur:

- The principal of the request is determined.
- The view function about to be accessed is looked up in the AccessRules mapping from view functions to access functions, returning the corresponding access function.
- That access function calls one or more data role predicates belonging to the request’s principal, which are evaluated based on the current principal’s implementation.
- The access rule function returns true or false. If true, the request executes as normal. If false, a 403 Access Denied error is returned.

### 5.3 Indivo API

The Indivo API provides an interface for Personal Health Applications to extend the functionality of the Indivo PCHR. The Indivo API abides by the REST design pattern: it is stateless, re-uses existing HTTP constructs such as caching, compression and content negotiation, and generally uses URLs to represent hierarchical resources, e.g. documents within a medical record.

This document provides an introduction to the API, with in-depth explanations of related core Indivo concepts where applicable. For a full listing of all available Indivo API calls, see [API Reference](#).

#### 5.3.1 Overview

Personal Health Applications (PHAs) make HTTP calls to the Indivo API endpoint using the REST convention. OAuth is used to authenticate all calls, either in 2-legged mode for simple authentication, or in 3-legged mode when the PHA is making a call with delegated authentication, i.e. on behalf of the user.

**Application Types**

We consider three types of applications:

- **User Applications**, which individual Indivo users can add to their record.
- **Administrative Applications**, which are used to perform account and record manipulations.
- **UI Applications**, which provide the public user interface to Indivo features.

Per Indivo installation, there is a small handful of UI and administrative applications, and quite a number of user applications.
Terminology

A record is the single set of medical information that pertains to an individual. It is composed of documents, including a demographics document which details the individual’s contact information and name. A record can be accessed by one or more accounts.

OAuth is the authentication protocol used by Indivo. In 2-legged OAuth, the PHA (the OAuth consumer) makes calls to Indivo (the OAuth service provider) using a consumer key to identify itself, and a consumer secret to sign the request. In 3-legged OAuth, the PHA makes calls to Indivo to access medical information as delegated by the user, using an additional token and token secret that pertain to the specific Indivo record being accessed.

5.3.2 Authentication

All calls to Indivo are authenticated using OAuth.

We detail the authentication process at Indivo Authentication.

5.3.3 Design Patterns

Some common design patterns are repeated throughout the API. Not all of these patterns are necessarily 100% supported by all Indivo API implementations, though when they are they are consistent.

Email Addresses as Identifiers

Core accounts in Indivo X are identified by email addresses, because email addresses provide mechanisms for distributed identification and messaging. When an email address is included in a URL, it must be URL encoded, where the @ sign turns into %40.

Paging/Filtering Results

When a list of results are returned, the URL ends in a / and the HTTP method is a GET, as is typical of REST design. In that case, Indivo X supports a generic query string that determines paging and ordering of the results:

?offset={offset}&limit={limit}&order_by={order_by}&status={document_status}&modified_since={modified_since}

- offset indicates which item number to start with, e.g. when getting a second batch of items.
- limit indicates the maximum number of items to return. This is used in combination with offset to accomplish paging.
- order_by is dependent on the fields returned in the list of items, and each call must thus define which fields are valid. Using an invalid field in order_by results in no effect on the output, as if order_by were absent.
- status can be used where applicable. It pertains to the status of documents and can currently be set to one of three options: ‘void’, ‘archived’ or ‘active’
- modified_since allows an application to look at items that have been modified since a given timestamp, so that incremental downloads may be possible.

Querying Results

As of the Beta3 release, calls that implement the basic paging operations above may also implement a more powerful query interface, also represented in the query string. In these cases (currently all of the minimal medical reports and the auditing calls), the following values may occur in the query string:
?offset={offset}&limit={limit}&order_by={order_by}&status={document_status}

These values function as before.

?group_by={group_field}&aggregate_by={aggregation_operator}*{aggregation_field}

group_by groups results by the specified field. It must be used in conjunction with aggregate_by, which aggregates the results by group, using the specified operation. If aggregate_by is passed without a group_by parameter, the aggregation is performed over the entire result set. Results that have been aggregated are returned in an aggregated format, not the typical reporting format.

?date_range={date_field}*{start_date}*{end_date}

date_range filters results and leaves only those with the specified field falling between start_date and end_date.

?date_group={date_field}*{time_increment}&?aggregate_by={aggregation_operator}*{aggregation_field}

date_group functions equivalently to group_by, except the groups are formed based on the values of the specified date field. For example, if the date field was ‘date_measured’, and the time increment was ‘month’, results would be returned grouped by the month of the date_measured field for each item. As with group_by, date_group must be used with an aggregator, and results are returned in an aggregated format.

?{FIELD}={VALUE}

This syntax adds additional filters to the query, returning only results having whose value for the property specified by ‘field’ matches ‘value’.

For each of these parameters, acceptable values for {field} are specified individually by the calls. A full listing of the minimal reporting fields, along with valid aggregation operators and date increments, may be found here.

External IDs

When a resource is created, the Indivo API offers the ability to create this resource using a PUT with an external_id in the URL, so that the call is idempotent: if a failure occurs, the call can be repeated safely and only the resource will not be created on the second call if it was already created successfully during the first call.

An external_id is only valid within a particular PHA scope. Other PHAs cannot see the external_id of a given document if they didn’t create the document, and certainly cannot access the document by external_id.

Some API calls which involve both creating documents and retrieving them, such as:

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/external/{APP_ID}/{EXTERNAL_ID}

For these calls, it can be confusing as to which document is referenced by an external id. In such cases, the following rule resolves confusion:

• The newly created document will always be assigned the passed external_id. The external_id will not be used to look up the existing document.

5.3.4 Managing Documents

Data stored in Indivo cannot by permanently deleted by default: the API enforces only appending data, not fully replacing it or removing it.
Reading Documents

GET /records/{RECORD_ID}/documents/,
GET /carenets/{CARENET_ID}/documents/  List documents within a record. Supports order by document metadata fields (see Indivo Document Metadata Schema).

The calls to GET /records/{RECORD_ID}/documents/ and GET /carenets/{CARENET_ID}/documents/ take a type querystring parameter, which filters the list of returned documents by their types.

A document’s type is (by default) the suffix of a URL that corresponds to the XML schema datatype, where the prefix is http://indivo.org/vocab/xml/documents#. Thus, type can be Medication, Lab, etc.

Indivo X supports storing XML documents whose datatype is not among the default Indivo X recommended types. In those cases, if the XML schema namespace doesn’t end in a / or #, then as is typical in the XML/RDF community, a # is used as delimiter in the URI. Examples of document types include:

- http://indivo.org/xml/phr/medication#Medication (Indivo 3.1 data type)
- urn:astm-org:CCR#ContinuityOfCareRecord, as per http://code.google.com/apis/health/ccrg_reference.html

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID},
GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID}  Fetch a single document.

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/meta,
GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID}/meta  Fetch metadata about a single document, using its internal or external id.

GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/versions/  List versions of a single document.

Writing Documents

PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/label,
PUT /records/{RECORD_ID}/documents/external/{APP_ID}/{EXTERNAL_ID}/label  Update a single document’s label.

POST /records/{RECORD_ID}/documents/,
PUT /records/{RECORD_ID}/documents/external/{APP_ID}/{EXTERNAL_ID}  Create a new document, and possibly assign it an external id.

Medical data cannot be replaced wholesale, only versioned. Thus, this call will fail (with a 400 Bad Request error code) if a document already exists in the given record with the given external ID.

POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace,
PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace/external/{APP_ID}/{EXTERNAL_ID}  Replace one document with a new document content. The existing document remains, but is marked suppressed and replaced by the new document.

Medical data cannot be replaced wholesale, only versioned. Thus, this call will fail (with a 400 Bad Request error code) if a document already exists in the given record with the given external ID.

Removing and Archiving Documents

Generally, documents in Indivo cannot be removed, they can only be versioned. However, mistakes happen, and Indivo must deal with these somehow. Also, information eventually is out of date or no longer relevant.

All such changes are encoded in the Indivo API as changes to document status.

POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/set-status  Change the status of a document. The passed status defines what happens to the specified document:
• **void:** If a document is entered in error, it can be marked as voided to indicate that the data is invalid.

  Only active documents can be voided. Voided documents are still reachable, but their metadata indicates their status, and by default they are not listed in typical document listings.

• **archived:** If a document is no longer relevant, it can be archived so that it doesn’t show up by default. Archival is different from voiding in that an archived document is still considered medically correct, just not particularly relevant anymore.

  Archived documents are still reachable, but their metadata indicates their archival status, and by default they are not listed in typical document listings.

• **active:** An active document is readily usable and will appear in search listings by default. Setting a document to active status will unvoid a voided document, or unarchive an archived document.

**GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/status-history**  
A document can be voided, unvoided, archived, unarchived any number of times. The status change applies to the entire version lineage of a document. The history of statuses, in reverse chronological order, can be obtained using this call.

### Relating Documents

It is often useful to relate documents, e.g. annotating a document, re-filling a prescription, connecting diagnoses to an encounter, etc. In Indivo X, these relations can be declared no matter the data type of the underlying document. An image of an X-ray might be related to an XML document that interprets it, but of course there is no room in the image file for a pointer. So all references are stored externally to the documents.

Relationship types are taken from a fixed list, including:

• interpretation
• annotation
• followup

Eventually, full URLs will be supported for relationship types. The fixed list of types will then correspond to http://indivo.org/vocab/documentrels#{rel_type}.

In the following calls, `{DOCUMENT_ID}` is the document being interpreted, and `{OTHER_DOCUMENT_ID}` or the POST content is the interpretation.

**PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/{OTHER_DOCUMENT_ID}**  
Create a new relationship of type REL_TYPE between the two passed documents.

**POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/, PUT /records/{RECORD_ID}**  
Create a new document and immediately relate it to an existing document, possibly assigning an external id to the newly created document.

  Medical data cannot be replaced wholesale, only versioned. Thus, this call will fail (with a 400 Bad Request error code) if a document already exists in the given record with the given external ID.

**GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/**  
List all documents related to the passed document by the relationship REL_TYPE.

  `{DOCUMENT_ID}` is the interpreted document, and the calls return all interpretations (that are of type REL_TYPE) of that document.

### Special Documents

Demographics documents are special in that there should only be one per record, and they should be easy to find.

See also:
Indivo X Documentation, Release 2.0.0

Indivo Document Demographics Schema  The XML Schema for Indivo Demographics Data

GET /records/{RECORD_ID}/demographics, GET /carenets/{CARENET_ID}/demographics
Fetch demographics from a carenet or record. Depending on the requested response_format return data is formatted as SMART RDF/XML (default), SDMJ, or SDMX.

PUT /records/{RECORD_ID}/demographics  Update or create demographics on a record.

5.3.5 Messaging and Notifications

Indivo supports a lightweight notification framework as well as a heavier message inbox. For more information, see Messaging and Notifications.

Messaging

GET /accounts/{ACCOUNT_ID}/inbox/  List available messages. By default, only non-archived messages are returned.

GET /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}  Fetch a single message.

POST /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}/archive  Archive a message.

POST /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}/accept
Accept a message attachment. A user can accept an attachment from a message into their medical record. This creates a new document on their record containing the contents of the attachment.

POST /accounts/{ACCOUNT_ID}/inbox/  Send a message to an account.

POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}  Send a message to a record. Messages to records can have attached documents (specified by the num_attachements parameter) which then need to be uploaded separately. The message isn’t delivered until all of its attachments are uploaded.

Since Accounts, not Records, are the users who log into the system to view messages, there is no way to view messages in a record’s inbox. Rather, when a message is sent to a record, every account authorized to view the message is sent a copy of the message, which they can retrieve via their account inbox.

POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}
Upload an attachment to a message.

Notifications

Notifications are intended to be a lightweight system for applications to alert users of activity in the application. This is especially relevant for apps that use sharing functionality: an app might want to notify other users of the app about a given user’s activity in it. UI apps should display these notifications in a twitter-feed like interface (our reference UI call it the ‘healthfeed’).

POST /records/{RECORD_ID}/notifications/  Send a notification to a record. As with inbox messages, notifications are propagated to the accounts that are authorized to view the record.

GET /accounts/{ACCOUNT_EMAIL}/notifications/  List available notifications.

5.3.6 Application-Specific Storage

Application-specific storage is meant for bookkeeping by individual applications that is not specific to any given record. These documents can be deleted, since they are not part of any permanent medical record. All application-specific storage API calls behave like the prior document API calls, though the documents are accessible only to the
application in question. Most implementations of the Indivo API will likely impose a quota on Applications to ensure they do not store large amounts of data in the application-specific storage. This quota may be application-specific, depending on what the application is approved to do.

Application-specific storage calls, since they don’t correspond to any given record, are all 2-legged oAuth calls.


GET /apps/{APP_ID}/documents/{DOCUMENT_ID} Fetch a single application-specific document.

GET /apps/{APP_ID}/documents/{DOCUMENT_ID}/meta, GET /apps/{APP_ID}/documents/external/{EXTERNAL_ID}/meta Fetch metadata about a single application-specific document, by its internal or external id.

POST /apps/{APP_ID}/documents/, PUT /apps/{APP_ID}/documents/external/{EXTERNAL_ID} Create an application-specific document, possibly assigning it an external id.

As this is application-level storage, making this call with an external id will overwrite any existing document with the same external id.

PUT /apps/{APP_ID}/documents/{DOCUMENT_ID}/label Update the label of an application-specific document.

DELETE /apps/{APP_ID}/documents/{DOCUMENT_ID} Delete an application-specific document. Since these documents do not contain medical data, deleting them is acceptable.

5.3.7 Record-Application-Specific Storage

Record-application-specific storage is meant for bookkeeping by individual applications. These documents can be deleted, since they are not part of the permanent medical record. All record-application-specific storage API calls behave like the prior document API calls, though the documents are accessible only to the application in question. Most implementations of the Indivo API will likely impose a quota on Applications to ensure they do not store large amounts of data in the record-application-specific storage. This quota may be application-specific, depending on what the application is approved to do.

Record-Application-specific storage calls are all 3-legged oAuth calls.


GET /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID} Fetch a single record-application-specific document.

GET /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID}/meta, GET /records/{RECORD_ID}/apps/{APP_ID}/documents/external/{EXTERNAL_ID}/meta Fetch metadata about a single record-application-specific document, by its internal or external id.

POST /records/{RECORD_ID}/apps/{APP_ID}/documents/, PUT /records/{RECORD_ID}/apps/{APP_ID}/documents/external/{EXTERNAL_ID} Create a record-application-specific document, possibly assigning it an external id.

As this is record-application-level storage, making this call with an external id will overwrite any existing document with the same external id.

PUT /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID}/label Update the label of a record-application-specific document.

DELETE /records/{RECORD_ID}/apps/{APP_ID}/documents/{DOCUMENT_ID} Delete a record-application-specific document. Since these documents do not contain medical data, deleting them is acceptable.
5.3.8 Processed Medical Reports

Indivo processes documents into medical reports. Each report can be altered by the basic paging mechanism or the more complex query interface described above. Over time, new reports may be introduced. For now, we define these as the minimal set of reports. Fields supported by individual reports for the querying interface may be found here. Response formats correspond to the Indivo Reporting Schema, and individual reports fit their individual datatype’s schema (see Medical Documents). If a report is accessed via a carenet, only documents that are shared into the carenet will appear in the results.

GET /records/{RECORD_ID}/reports/minimal/equipment/, GET /carenets/{CARENET_ID}/reports/minimal/equipment/ List equipment for a given record.

GET /records/{RECORD_ID}/reports/minimal/labs/, GET /carenets/{CARENET_ID}/reports/minimal/labs/ List lab results for a given record.

GET /records/{RECORD_ID}/reports/minimal/measurements/{LAB_CODE}/, GET /carenets/{CARENET_ID}/reports/minimal/measurements/{LAB_CODE}/ List measurements for a given record.

GET /records/{RECORD_ID}/reports/minimal/procedures/, GET /carenets/{CARENET_ID}/reports/minimal/procedures/ List procedures for a given record.

GET /records/{RECORD_ID}/reports/minimal/simple-clinical-notes/, GET /carenets/{CARENET_ID}/reports/minimal/simple-clinical-notes/ List clinical notes for a given record.

5.3.9 SMART API Calls

As Indivo now supports the SMART API, the following calls are now available:

GET /records/{RECORD_ID}/(MODEL_NAME)/ Get a SMART RDF list of a patient’s medical data of type MODEL_NAME. Available data models are:

- allergies
- encounters
- fulfillments
- immunizations
- lab_results
- medications
- problems
- vital_signs

GET /apps/{APP_ID}/manifest, GET /apps/manifests/ Get SMART-style JSON manifests for one or all apps registered with this instance of Indivo.

GET /ontology Get the ontology used by a SMART container

GET /capabilities/ Get the SMART capabilities for this instance of Indivo.

GET /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences Get account preferences for a specific application.

PUT /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences Set account preferences for a specific application. Overrides previous preferences.

DELETE /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences Remove all account preferences for a specific application.
5.3.10 Generic Reports

Indivo provides the ability to run ‘generic’ reports over all *data models*. These reports support the *API Query Interface*, and provide an out of the box solution for reporting over core and contributed data models, with the possibility for *customization*.

```
GET /records/{RECORD_ID}/reports/{DATA_MODEL}/, GET /carenets/{CARENET_ID}/reports/{DATA_MODEL}/
```

List a patient’s medical data.

5.3.11 Coding Systems

A number of Indivo documents contain coded values. These can be based on UMLS, SNOMED, etc. Indivo provides a generic API for looking up coded values. This API is particularly built to support live autocomplete in JavaScript.

```
GET /codes/systems/  List available coding systems. Return data is in JSON format.
GET /codes/systems/{SHORT_NAME}/query  Search a coding system for a value.
```

5.3.12 Autonomous Apps API

Autonomous user applications are unlike standard user apps in that they may not have a user interface, and require access to records without an active user session. In order to authenticate against Indivo on behalf of records at any time, autonomous apps may make the following calls:

```
GET /apps/{APP_ID}/records/  Return a list of records which have enabled the app, and to which (therefore) the app can authenticate and acquire access.
POST /apps/{APP_ID}/records/{RECORD_ID}/access_token  Retrieve a valid access token providing the app with access to a record. This call will only succeed if the app is autonomous, and if the record has enabled the app.
```

Using this call, an autonomous app can retrieve a valid access token for any record on which it is enabled, without an active user session.

5.3.13 Administrative API

Admin applications have access to Indivo’s administrative API, which enables control and setup of records and accounts.

**Account Administration**

```
GET /accounts/{ACCOUNT_ID}  Get information about an account. The account_id must be in the form of an email address.
POST /accounts/{ACCOUNT_ID}/info-set  Set information about an account, including the name and contact email of the account holder.
GET /accounts/search  Search for accounts by name or contact email.
GET /accounts/{ACCOUNT_EMAIL}/records/  List available records on an account. Supports order by label.
POST /accounts/  Create an account.
```

The primary and secondary secret arguments are optional and are used for helping the user initialize their account securely. A primary secret is sent directly by Indivo X server to the user at their *ACCOUNT_ID* email address in
the form of a URL with an embedded secret. A secondary secret is generated by Indivo X and made available to the admin application using the GET /accounts/{ACCOUNT_ID}/secret call for the account. If it is asked for in this call, it is required at account activation time right after the user clicks on the activation URL (aka the primary secret). A secondary secret makes sense only if a primary secret is also requested. That’s why it’s called “secondary.”

**POST /accounts/{ACCOUNT_ID}/authsystems/** Add an authentication system to an account.

Accounts initially have no “authentication systems” attached to them. Over time, Indivo accounts will be usable with OpenID and other authentication systems. An account needs to enabled for each authentication system that we want to use for that account. The default system is “password”. Thus, this call, when used with the “password” system, will set up the password and username for a new user.

**POST /accounts/{ACCOUNT_ID}/secret-resend** Resend an account’s initialization URL (which contains the primary secret for the account). This is useful if the account holder loses the original email.

**POST /accounts/{ACCOUNT_ID}/forgot-password** Reset an account when its password is forgotten.

If a password is forgotten, the solution is to reset the account and email the user as with their initialization email. This will prevent logins until the new initialization URL is clicked, and the new password is entered.

This could be accomplished with separate calls to POST /accounts/{ACCOUNT_ID}/reset, which sets the account state to uninitialized and resets the account secrets, and POST /accounts/{ACCOUNT_ID}/secret-resend, but this call combines both actions.

Note that this call resets both the primary and secondary secrets. The user will need to be given this secondary secret in a channel other than email. If a User Interface Application performed this reset, then the secondary secret should display on screen while the primary secret is automatically sent by email. The user interface could obtain the secondary secret (which is short) by calling GET /accounts/{ACCOUNT_ID}/secret, but the call to POST /accounts/{ACCOUNT_ID}/forgot-password returns the secondary secret to avoid the extra call.

**POST /accounts/{ACCOUNT_ID}/initialize/{PRIMARY_SECRET}** Initialize a new account.

Initializing an account that has been reset requires both the primary and secondary secrets. The primary secret is sent in the URL, and the secondary secret should be collected by the user interface. Specifically, the recommended process is:

- Indivo Backend server sends the reinitialization URL to the user as:

```
INDIVO_UI_APP_LOCATION/account/initialize/account_id/primary_secret
```

- An Indivo UI App checks that the requested account is indeed in uninitialized state and prompts the user for his secondary secret (which the user knows simply as the “secret”) and his desired username and password.

- The Indivo UI App initializes the account using this call.

- The Indivo UI app sets up the account with the built-in password authsystem using the username/password provided by the user and the API call POST /accounts/{ACCOUNT_ID}/authsystems/.

**POST /accounts/{ACCOUNT_ID}/set-state** Set an account’s state. Possible account states are:

- uninitialized: an account that has been created by an administrative application and has not been activated by the user yet (with their confirmation URL and code).

- active: a normal active account.

- disabled: an account locked because of too many failed login attempts.

- retired: an account that is no longer in use.

**POST /accounts/{ACCOUNT_ID}/authsystems/password/set** Force an account’s password to a new value. This should be used only in the context of an account reinitialization.
POST /accounts/{ACCOUNT_ID}/authsystems/password/change Allow a user to change an account password. The given old password must be correct for this change to succeed. This is a 3-legged call, since the user is the one driving this interaction (unlike POST /accounts/{ACCOUNT_ID}/authsystems/password/set, wherein the admin app is forcefully setting a password).

GET /accounts/{ACCOUNT_ID}/primary-secret Fetch an account’s primary secret. This should be used very sparingly as the primary secret should rarely be seen outside of the Indivo backend.

Record Administration

GET /records/{RECORD_ID} Get info about a single record.

GET /records/search Search Indivo for existing records by record label.

GET /records/{RECORD_ID}/owner Get the owner of a record

GET /records/{RECORD_ID}/apps/ List applications attached to a record. Supports order by name.

POST /records/, PUT /records/external/{APP_ID}/{EXTERNAL_ID} Create a new record, possibly assigning it an external id. This call requires a valid Indivo Demographics Document in order to create the record.

PUT /records/{RECORD_ID}/owner Set the owner of a record.

POST /records/{RECORD_ID}/apps/{APP_ID}/setup Prime a record with a user app. This sets up an app to run against a record without user consent. It should be used only in cases where obtaining consent is impossible or unnecessary (i.e., at a hospital installation of Indivo, this call could be used to prime all new records with the syncher application that pulls data into Indivo from the hospital EMR).

PUT /records/{RECORD_ID}/apps/{APP_ID} Enable an app to run against a record. This gives the app access to the entire record.

DELETE /records/{RECORD_ID}/apps/{APP_ID} Remove a user app from a record.

5.3.14 Indivo Chrome / User Interface API

These API calls are reserved for the UI server, which is deeply trusted to authorized other applications, proxy the user’s credentials, etc. It’s only a separate server for modularity, otherwise it has the same level of trust as the backend Indivo server.

POST /oauth/internal/session_create Create a web-session for a user. This call returns a session token that can be used to authenticate 3-legged calls on behalf of the user for the duration of a standard web session (30 minutes by default)

POST /oauth/internal/request_tokens/{REQUEST_TOKEN}/claim Claim a request token on behalf of a user. Before a request token can be viewed at all, it has to be claimed by a user. This ensures that a request token can’t be partially used by one user and completed by another.

The session-based chrome authentication will indicate to the backend which Account to associate with this request token. Once this call has been made for a request token, a second call with different session authentication will fail. (A second call with the same user authentication will be just fine, we don’t want a reload to cause a problem.)

If the request token is bound to an Indivo record (because the PHA knew it was authorizing for a given record), and the claimant does not have administrative rights over the record, this call will fail and the request token will be invalidated.
GET /oauth/internal/request_tokens/{REQUEST_TOKEN}/info Retrieve information about an OAuth request token.

When authorizing a request token, the Indivo UI needs to know what that token represents. Once the token is claimed, the request token yields information via this call.

This call can only be called with session authentication matching the Account which claimed the request token earlier.

If a record_id is present in the response data, then the kind element is also present and indicates:

- **new**: a new request for a PHA that has not been authorized for this record yet
- **same**: a request for a PHA that is already attached to the record and no new permissions are requested
- **upgrade**: a request for a PHA that is already attached to the record but that is asking for more permissions or more permissive usage of the data.

In the same case, the Chrome UI is allowed to immediately approve the request token. In other cases, the Chrome UI must explain to the user that new permissions or rights are being granted and prompt the user for approval.

POST /oauth/internal/request_tokens/{REQUEST_TOKEN}/approve Approve a request token on behalf of a user.

If a user approves an app addition, then the Chrome UI server needs to let the backend know.

This call, if it succeeds with a 200 OK, will return the location to which the user’s browser should be redirected:

location={url_to_redirect_to}

This call’s session authentication must match that which claimed the request token. The record_id is the record to which the user is attaching the application (i.e. my child’s record, not my own.) If the request token was pre-bound to a record, this record_id parameter must match, or this will throw an error.

POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials Get credentials for Connect-style Authentication for a user app, and authorize them on behalf of an account.

This call will return tokens that can be used to sign future API calls by the user app, proxied by the UI.

GET /accounts/{ACCOUNT_ID}/check-secrets/{PRIMARY_SECRET} Check the primary and secondary secrets of an account.

GET /oauth/internal/surl-verify Verify a signed URL.

In some cases, an Indivo app will sign a URL that directs the user to the Indivo UI. A prime example is the use of Indivo Chrome widgets, i.e. the Document Sharing widget, that apps can embed within their user interface to reuse functionality from Indivo Chrome. A signed URL looks like this:

/widgets/WidgetName?param1=foo&param2=bar&surl_timestamp={TIMESTAMP}&surl_token={TOKEN}&surl_sig={SIGNATURE}

The signature contained in surl_sig is effectively a signature on the rest of the URL. The signature algorithm is as follows:

1. An app, with OAuth access token TOKEN and OAuth access token secret SECRET, wishes to sign a URL.
2. The app generates the SURL secret that corresponds to this access token as follows:
   
   $$<SURL_SECRET> = HMAC(<TOKEN_SECRET>, "SURL-SECRET")$$

   using base64 encoding, where the idea is to actually sign the string “SURL-SECRET” to obtain the SURL secret itself.
3. this SURL secret is then used to sign the URL, first by appending a timestamp, the SURL token, and then computing the signature:

\[
<\text{SURL\_SIG}> = \text{HMAC}(<\text{SURL\_SECRET}>, \\
"/widgets/WidgeName?...&surl\_timestamp=<\text{TIMESTAMP}>&surl\_token=<\text{TOKEN}>"
\]

in base 64, then appending it as a query parameter surl_sig.

### 5.3.15 Sharing

#### Overview

We want to simplify sharing. Indivo has two main mechanisms for sharing patient records with other accounts: *Full Shares* and *Carenets*.

**Full Share** A user may choose to share the entirety of their record with another account. The recipient account will then have access to all data (past, present, and future) contained in the record, and will be able to run any apps that have been bound to the record. The recipient of a full share will also be able to add new applications to the record and run them against data in the record.

Similarly, a user may choose to add an application to their full record. This effectively creates a ‘full share’ of the record with that application: the app has access to all data in the record.

As an example, a teen user of Indivo might choose to set up a full share of his/her record with a parent or guardian.

**Carenet** Full shares are not very flexible: they are an all or nothing proposition. In cases where sharing data appropriately requires more granularity or complexity, Indivo provides *carenets*, which allow a record to specify groups of *accounts* and *apps* that all have transparent access to whatever *data* the record shares into the carenet.

By default, each record will have 3 simple carenets: physicians, family, and work/school.

As an example, a patient might create an ‘exercise’ carenet, into which they place:

- *data*: blood-pressure readings, pedometer output, and other data associated with maintaining a healthy lifestyle.
- *apps*: blood-pressure viewers, exercise-trackers, and other apps that help the patient organize and interact with their exercise data.
- *accounts*: The patient’s Primary Care Physician, personal trainer, friends, or any other person with an interest in helping the patient develop healthy exercise habits.

Now anyone on the *accounts* list can log into Indivo and run any app on the *apps* list against any data on the *data* list.

Data can be placed into carenets individually, or autoshared by Document type. Users can override the type-auto-sharing on a document-by-document basis.

**Autoshare** Documents of a certain type can be auto-shared, so that they are added to a carenet automatically when they are added to the record. When auto-share preferences are set for a type of document within a given carenet, these preferences apply to all documents that do not have an explicit sharing preference declared on them.

**Nevershare** A user should be able to ask that a specific document be “never shared”. This flag prevents any sharing, no matter what the auto-share rules may be.

#### Authorization into a CareNet

When an app is added, it is normally given, along with its oAuth token, an *xoauth_indivo_record_id* that the token corresponds to. If the app is added to a carenet instead of a record, the app will receive instead an
Carenets are useless until data has been shared into them. Data can be shared explicitly at the granularity of individual documents, or implicitly at the granularity of document type.

**Data in Carenets**

Carenets are useless until data has been shared into them. Data can be shared explicitly at the granularity of individual documents, or implicitly at the granularity of document type.

**PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}** Place a document in a carenet.

When a document is explicitly shared with a carenet, it is no longer tied to the auto-sharing rules for that carenet. However, auto-sharing rules with other carenets still apply.

**DELETE /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}** Remove a document from a carenet

When a document is explicitly UNshared from a carenet, it is no longer tied to the auto-sharing rules for that carenet. However, auto-sharing rules with other carenets still apply.
GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/ List carenets where a document is present

The **mode** attribute indicates how this document is shared. **explicit** means that the sharing preferences for this document are explicitly set. **bytype** indicates that it was auto-shared by document type. Other modes may be enabled in the future.

The **value** attribute indicates a negative share with a carenet, meaning that the user explicitly wants this document not shared with this carenet, even if auto-share rules would otherwise share it. Obviously this only makes sense for explicit carenet-shares.

**POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}/autoshare-revert**
Revert a document to auto-share rules. This means that, for this carenet, this document reverts to automatic sharing rules. This might mean a removal of the share with this carenet, or an addition, or no effect. However, from this point on, the record-wide rules apply.

<table>
<thead>
<tr>
<th>Warning: This call has not yet been implemented.</th>
</tr>
</thead>
</table>

GET /records/{RECORD_ID}/autoshare/bytype/ List auto-sharing preferences for a given document type with a record. This call returns a list of carenets into which the document type is auto-shared.

GET /records/{RECORD_ID}/autoshare/bytype/all List all auto-sharing preferences for a record. This call returns a list of document types with the carenets into which each type is auto-shared.

**POST /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/set** Add an auto-share of a given document type into a given carenet. This share applies to all documents that do not have an explicit sharing preference declared on them.

**POST /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/unset** Remove an auto-share for a given document type from a given carenet.

**PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare** Set the never share flag on a document. A user should be able to ask that a document be “never shared”. This flag prevents any sharing, no matter what the auto-share rules may be.

**DELETE /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare** Remove the never-share flag on a document.

### Apps in Carenets

Users needs to be able to place apps inside carenets in addition to documents, so that other accounts can run the applications. There are no issues with read/write permissions here, as permissions are associated with the accounts in a carenet, not the apps.

**GET /carenets/{CARENET_ID}/apps/** List all apps in a carenet.

**PUT /carenets/{CARENET_ID}/apps/{PHA_EMAIL}** Add an app to a carenet.

**DELETE /carenets/{CARENET_ID}/apps/{PHA_EMAIL}** Remove an app from a carenet.

**GET /carenets/{CARENET_ID}/apps/{APP_ID}/permissions** Check an app’s permissions within a carenet. Since permissions are currently handled on accounts, not apps, this call will always indicate that the app has full permissions on the carenet.

| Warning: This call has not yet been implemented. |
Accounts in Carenets

Users need to be able to place other accounts inside carenets so that they can share data and apps. When accounts are added to a carenet, they are assigned read/write permissions, which define what actions they can take on data in the carenet.

GET /carenets/{CARENET_ID}/accounts/ List all accounts in a carenet.

POST /carenets/{CARENET_ID}/accounts/ Add an account to a carenet.

DELETE /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID} Remove an Account from a carenet.

GET /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID}/permissions Check an account’s permissions within a given carenet. This call will return a list of document types, and whether the account may write to each one within the given carenet.

or now, the document type is always “*”, since read/write permissioning is not currently more granular than at the carenet level. We may eventually permit permissioning by document types within a carenet, in which case this all will be more informative.

Building Carenet-aware Apps

Indivo apps are given a record_id and an access token that matches that record to read and write documents, read reports, annotate, etc. In a sharing scenario, apps must become carenet-aware.

Requirements

- An app should be easily placed within any number of carenets, i.e. physicians and family, but not work/school.
- When an app is activated on a given record, it must have access to no more data than the user who activated it. For example, if the owner selects the app, then the app may have access to the entire record. If the owner’s school nurse activates the app, the nurse should have access to only the data that is in the work/school carenet.
- There may be a need to further constrain an app, so that even if the owner activates the app, it should not be able to see every data type, or may be constrained to one of the carenets anyways. This is deferred for now.
- We must not depend on app developers to properly partition information. If an app is active in both the Family and Physicians carenets, and knows that the record_id is the same in both cases, it may well intermix data without realizing it. This would be bad. We need to make it harder for apps to hurt the user.

Scenario

Alice owns her Indivo record and has shared it with Bob, her HR representative at work, placing Bob in the “Work/School” carenet. Alice is pregnant but does not wish to reveal this information to her co-workers just yet. She has added the “Pregnancy Tracker” app to her record, making it visible to the Family carenet. Alice has a history of depression, information which she has shared with her Physicians, but not with her Family.

Visible Apps

The “Pregnancy Tracker” app has been added to the Family and Physicians carenets, but not the Work/School carenet, so Bob cannot even see the application when he visits Alice’s record. This is enforced by the Indivo platform itself.

Activating and using an App

Charlie, Alice’s father, is eager to check up on his future grandchild’s progress. He logs into Indivo, selects Alice’s record. He sees “Pregnancy Tracker” because that app is visible to the Family carenet. He launches the app, and uses
its functionality to track Alice’s progress, her fetus’s growth, her blood tests, etc. The process when launching the app is:

- Clicking on the app directs the IFRAME to the start_url for the pregnancy tracker. The app must receive an indication of which record is being accessed at this point. This cannot be the record_id alone, and we may not even want to include the record_id at all, otherwise the app might confuse this data with that accessible to Physicians later on. Thus, instead of passing record_id to the IFRAME, Indivo passes only carenet_id.
- The oAuth process begins with the carenet_id only as part of the request for a request token.
- Indivo checks that the logged-in-user has the right to access this carenet, and if so authorizes the token.
- The token is bound to that carenet only, and cannot be used on any other carenet.
- The app can make requests to
  
  /carenets/(carenet_id)/documents/...

  without using the record_id at all. It doesn’t need to know the record_id.
- When the app is later activated by a Physician, who does have access to Alice’s history of depression, the app gets a different carenet_id, and from that carenet has access to the documents including mental health.
- This is not fool-proof: we still probably need to give the app access to some record information that will yield a unique identifier using the name, DoB, etc... but at least the default behavior for the app will not allow error-prone tracking across carenets.

**oAuth Mechanics**

We start with:

- A CarenetAccount row that shares a record’s carenet with another account
- A Share row that indicates that an app has access to the record
- A CarenetPHA row that makes the app available in the carenet.

The oAuth process is then:

- PHA requests a request token with a carenet_id instead of a record_id as parameter.
- PHA needs to have a share into the record or into the specific carenet for this to succeed.
- The request token needs to keep track of the carenet, because the Share might be for the whole record.
- The user approving the request token should be in the carenet in question.
- The access token already stores the account of the person it’s proxying for, so that should be enough.

### 5.3.16 Auditing

As Indivo will be installed at HIPAA-compliant hospital sites, it is important that it be able to track system usage. The auditing system logs all incoming requests to Indivo that use the Indivo API. To learn more about auditing in Indivo, see the audit system’s documentation.

**GET /records/{RECORD_ID}/audits/query/** Query the audit system. This call allows for general queries over the audit logs. The query is specified in the parameters via the **API Query Interface**, and returns results in the style of Medical Reports.

All subsequent calls are deprecated, but maintained (for now) for backwards compatibility.
GET /records/{RECORD_ID}/audits/  List all audit log entries where a given record was accessed.
   Deprecated since version 1.0.0.

GET /records/{RECORD_ID}/audits/documents/{DOCUMENT_ID}/  List all audit log entries where a given document was accessed.
   Deprecated since version 1.0.0.

GET /records/{RECORD_ID}/audits/documents/{DOCUMENT_ID}/functions/{FUNCTION_NAME}/  List all audit log entries where a given document was accessed via a given internal Django view function.
   Deprecated since version 1.0.0.

5.4 The Audit System

Auditing will live in a middleware that is called after the Authentication and Authorization middleware.
The ordering is so that auditing will know the principal and view_func/view_(kw)args during processing.
When no principal is present we will not audit.
If principal is not None then we will record relevant information related to the request.
The amount of information audited and which calls to audit will be configurable in the top-level settings.py file

5.4.1 Data to Audit

Redundancy is built into the audit table so that as few assumptions are made when investigating a req/resp. as possible.
The proposed table will be broken down into five sections:

Basic Info

This information will always be audited, and contains basic information about the request.

• datetime: When the request was made
• view_func: Which Indivo View Function was called by the request
• request_successful: Was the request successful, or were there errors in its execution

Principal Info

Information about the principal of the request.

• effective_principal_email: The email of the principal making the request
• proxied_by_email: The email of the principal proxied by the principal making the request (i.e., the email of the Account being proxied by a PHA)

Resources

Information about the resources being accessed by a request

• carenet_id: Identifies the carenet a request is made through
• record_id: Identifies the record a request is made through
• pha_id: Identifies the PHA a request modifies
• document_id: Identifies the document accessed by a request
• external_id: Identifies the external identifier used to represent a resource in a request
• message_id: Identifies the message accessed by a request

Request Info

Information carried in by the request

• req_url: The URL to which the request was made
• req_ip_address: The ip address from which the request originated
• req_domain: The domain from which the request originated
• req_headers: Headers associated with the request, including OAuth credentials
• req_method: The HTTP method with which the request was sent (GET, PUT, POST, DELETE)

Response Info

Information about the response being returned

• resp_code: The HTTP response code returned by the request
• resp_headers: Headers sent out with the response, including content type

5.4.2 Configuring the Audit System

The following fields in settings.py may be altered to configure the system:

• AUDIT_LEVEL: Values are HIGH, MED, LOW, NONE. Controls the amount of information recorded for each call. See Audit Levels below for details.
• AUDIT_OAUTH: Values are True, False. Controls whether or not calls related to the OAuth dance are audited.
• AUDIT_FAILURE: Values are True, False. Controls whether or not calls which exited with failure (HTTP Response Code of 400 or greater) are audited.

Default Values

Defaults for the audit settings record as much information as possible. They are:

• AUDIT_LEVEL: HIGH
• AUDIT_OAUTH: True
• AUDIT_FAILURE: True

Audit Levels

The information recorded at each audit level is also configurable. Defaults are:

• NONE: no information is recorded.
• LOW: Basic Information and Principal Information are recorded.
• **MED**: Basic Information, Principal Information, and Resources are recorded.
• **HIGH**: Basic Information, Principal Information, Resources, Request Information, and Response information are recorded.

### 5.4.3 Querying the Audit System

There will be a new API call that implements the API **Query Interface**. Calls to:

GET /records/{record_id}/audits/query/

May retrieve audit entries filtered and grouped by the following fields:

- **document_id**: The document modified by the request. **String**
- **external_id**: The external id used to reference a resource in the request. **String**
- **request_date**: The date on which the request was made. **Date**
- **function_name**: The internal Indivo X view function called by the request. **String**
- **principal_email**: The email of the principal making the request. **String**
- **proxied_by_email**: The email of the principal proxied by the principal making the request (i.e., the email of the Account being proxied by a PHA). **String**

The default ordering on results will be in descending order by **request_date**.

Audit queries will be returned (like all data returned form the API Query Interface) according to the **Indivo Reporting Schema**. Individual items will validate against the **Indivo Audit Log Schema**.

### 5.4.4 Compatibility Issues with old Audit System

The Beta1 interface calls will be preserved, although they are now deprecated.

However, in order to accommodate the new data being stored, the output schema for audit logs has changed. Audit logs will now be returned according to the **Indivo Reporting Schema** and new **Indivo Audit Log Schema**, as with the new call above.

### 5.5 Indivo Authentication

As of version 2.0, Indivo now permits three methods of authentication:

- **Traditional oAuth**
- **In-browser Connect Authentication** (a la. SMART CONNECT)
- **Pre-generated REST Authentication** (a la. SMART REST)

#### 5.5.1 Indivo oAuth

Here, we provide details of exactly how Indivo uses oAuth, in particular the options that are supported and those that aren’t.
Specifics of our OAuth implementation

OAuth Section 5.2 defines three possible approaches to sending OAuth Protocol Parameters. In Indivo, we use exclusively the HTTP authorization header, as defined by the OAuth specification, as the preferred method. We allow `oauth_callback` and `oauth_verifier` to be provided as POST parameters, only because some client libraries that are not OAuth 1.0a-compatible cannot otherwise connect.

We require all changes suggested by the OAuth 1.0a revision:

- When asking for a request_token, the PHA must provide `oauth_callback` as an extra OAuth parameter. This can be a URL where Indivo X will send the user after token authorization, or it can be `oob` to use the default URL specified by the PHA at registration time.
- When responding to a request token, Indivo X will include `oauth_callback_confirmed=true`
- A PHA's callback URL should accept both `oauth_token` and `oauth_verifier` parameters.
- When exchanging a request token for an access token, the PHA must provide the corresponding `oauth_verifier` parameter in addition to all existing parameters.

In addition, we implement the following constraints:

- As per the OAuth recommended approach, all OAuth token setup and exchange calls use the POST method. The Indivo server will respond with a 405 Method Not Allowed error code to any GET request against its OAuth protocol URLs.
- The `oauth_version` parameter is mandatory. Every PHA request should include the `oauth_version` parameter. The only supported value in Indivo at this point is `oauth_version=1.0`.
- Given an Indivo Server running at `https://INDIVO_SERVER`, the OAuth URLs are defined as follows:
  - Request Token URL: `https://INDIVO_SERVER/oauth/request_token`
  - User Authorization URL: Since there is a UI component to enabling User Authorization (i.e., we have to obtain their consent), Indivo_Server does not explicitly offer a User Authorization URL via its API. Individual UI apps can (with a valid user session) authorize tokens on behalf of users with a call to `POST /oauth/internal/request_tokens/{TOKEN}/approve`, but each individual UI-app implementation will provide a different app-facing User Authorization URL. We recommend using `https://UI_SERVER/oauth/authorize?oauth_token={token}`, which is the URL used by our reference UI-app implementation.
  - Access Token URL: `https://INDIVO_SERVER/oauth/access_token`
- OAuth defines a number of default signature methods and leaves open the possibility of using other signature methods. Indivo supports only one request signature method: **HMAC-SHA1**. Support for RSA-SHA1 may eventually be added.

Body and Content-Type

By default, OAuth only signs the body of HTTP requests that are form-url-encoded. Indivo uses the OAuth Body Hash Extension to ensure that raw POSTs, e.g. to send XML documents, become part of the signature. When the body-hash extension is activated, Indivo also expects an additional parameter, `oauth_content_type`, to certify the content type HTTP header (and prevent content-sniffing attacks.)
User Applications (PHAs)

PHA Registration

A PHA registers with Indivo using an Indivo-installation specific process, at the conclusion of which both Indivo and the PHA agree on:

- The name of the PHA, e.g. “Medical Surveys”
- An OAuth consumer_key and consumer_secret.
- A start_url_template for the PHA, e.g. http://acme.com/indivoapp?record_id={indivo_record_id}&document_id={document_id}
- A callback_url for the PHA, e.g. http://acme.com/success_after_indivo, which should expect to receive query parameters oauth_token and oauth_verifier.
- Whether the PHA has a web user interface (certain applications that synchronize data have no UI), and whether that PHA is frameable inside Indivo.
- Whether the PHA is autonomous or not, and if it is, why it wants that kind of access to the user’s personal health record.

IMPORTANTLY, the callback_url is the only URL that Indivo will return the user to after a successful PHA attachment. Indivo does not support a custom OAuth callback URL.

Autonomous Apps

An autonomous app is one that wants to access the user's record while the user is not connected. PHAs that qualify include hospital data connectors, drug-interaction checkers, etc. There are very good reasons for PHAs to access a record while the user is not online, but we want to ensure that users understand the implications, and thus the Indivo authorization pathway looks different depending on whether an app is autonomous or not.

An app must choose to be autonomous at registration time. It must be autonomous for all users, or for none.

An autonomous app accesses the entire record by default, and the user must consent to this. This design choice is meant to prevent medical mistakes for automated apps that, for example, check for drug-drug interactions but may fail to notify the user if they have only partial data access. An autonomous app thus triggers the appropriate authorization screen that warns the user about the long-term, autonomous access, displays the app’s reason for requesting this type of access, and simply gives the user a yes/no choice.

Autonomous apps can, in some circumstances, have no user-interface. This might happen if, for example, a hospital connector application sits behind the hospital firewall and connects autonomously to the Indivo record to upload hospital data into the PCHR, but never lets the user connect directly to the app itself. There are two ways, currently, to authorize such an application. The first is via admin-based PHA setup, where an administrative app primes the Indivo record with this app. Alternatively, the UI may allow users to permission autonomous apps without an interface. In this case, there is no oauth dance: the user ‘enables’ the app, and the app is then able to acquire access tokens for enabled users directly. In any case, these apps must declare their lack of UI at registration time, much like they declare their being autonomous or not. Only autonomous apps can choose to forgo a UI.

A non-autonomous app, on the other hand, is one that is meant to be used by whoever is logged in and has access to the record in question. Depending on which user has launched the app, the app’s permissions might differ. For example, when Alice uses the Problems App within her record, she should see “all” of her problems. However, when Bob, her co-worker, uses the Problems App to view Alice’s record, he should see only those problems which Alice has chosen to let him see. Thus, a non-autonomous app exists purely to proxy a human user’s clicks and perform some visualization / data entry assistance functionality. Non-autonomous apps are thus constrained to a carenet at the time that the user clicks on the app name to launch it. When Bob launches the Problems App on Alice’s record, the Problems App receives an access token that is constrained to Alice’s “Work” carenet, and the app can only access the
problems Alice has made available within her Work carenet. All access tokens for non-autonomous apps are valid only for the duration of a web session.

**Connecting a PHA to a Record**

A user opts to add a PHA to her Indivo record by enabling it in the UI. At any subsequent point, when the user attempts to use the PHA (i.e. by clicking on it in the UI), she is sent to the PHA’s `start URL` with the `indivo_record_id` filled in. The PHA may present informational content if it so desires, then is expected to begin the OAuth authorization process. When the PHA begins the OAuth process, it should do so with the indicated `indivo_record_id` that it received when its `start_url` was accessed.

**Obtain a Request Token**

A PHA begins its access request for a user when the user visits the PHA’s `start URL`. While the user’s browser awaits a response, the PHA obtains from the Indivo Server a request token. This is accomplished by issuing a signed POST 2-legged OAuth request to the Request Token URL:

```
https://INDIVO_SERVER/oauth/request_token
```

with optional form parameter `indivo_record_id`. Again, if the PHA was accessed via its `start_url` with the Indivo record ID filled in, it should use this record ID at this point in obtaining the request token. Otherwise, the user interface will be thoroughly confusing.

This call returns an OAuth token:

```
oauth_token={token}&oauth_token_secret={secret}
```

The PHA is expected to store the Request Token and its correspondence to this specific user, likely in the web session.

**Authorize the Request Token**

Once it has obtained a request token, with the user’s browser still waiting for a response, the PHA responds by redirecting the user’s browser to the User Authorization URL on an Indivo UI app, indicated in the request token response above, or by default:

```
https://UI_SERVER/oauth/authorize?oauth_token=<REQUEST_TOKEN>
```

with the `request_token` as a URL query parameter named `oauth_token`. Note how this URL is not a signed OAuth request. This step is simply a redirection of the user’s browser to her Indivo account in order to prompt for and obtain authorization.

Indivo prompts the user to authenticate if she isn’t already logged in. Indivo then associates the request token with this user, and only this user can proceed with this specific request token. It is an error for a PHA to attempt to reuse request tokens, and Indivo will prevent this from happening.

Indivo then presents the user with the details of the PHA’s requested permissions.

The user can choose to cancel the process, in which case no further requests are issued, the PHA is not notified, and the request token is discarded.

**Obtain an Access Token**

If the user agrees to connect with the PHA, Indivo redirects the user browser to the PHA’s `callback_url`, as specified by the PHA at registration time. Appended to this `callback_url` are the `oauth_token`, the request token that identifies this authorization dance, and the `oauth_verifier`. The PHA is encouraged to check that the `oauth_token` matches the token stored in its web-session.

The PHA must now exchange the Request Token for an Access Token. This is accomplished using a 3-legged OAuth POST request, with the request token and secret, to:
In response to this request, the PHA obtains an Access Token, including one of two optional parameters:

oauth_token=<TOKEN>&oauth_token_secret=<SECRET>&xoauth_indivo_record_id=<RECORD_ID>

or

oauth_token=<TOKEN>&oauth_token_secret=<SECRET>&xoauth_indivo_carenet_id=<CARENET_ID>

This token can then be used by the PHA to make 3-legged OAuth calls to Indivo. The Indivo record ID parameter indicates which record this token is bound to, while the carenet indicates which portion of the system the PHA can access.

**Interact and Re-Auth** At this point, the PHA has an access token, an access secret, an Indivo record ID, and an Indivo privacy group. These credentials allow the PHA to make calls to the Indivo Server to obtain data from the given Indivo record. If the PHA provides a direct web interface to the user, this UI is delivered inside an IFRAME within the Indivo User Interface.

A few days later, when the user returns to his Indivo record, he can click on any of the PHAs he has already authorized. The PHA, however, does not know immediately who this user is. To communicate the user’s identity to the PHA, Indivo simply re-performs the OAuth dance, setting the IFRAME’s URL to the PHA’s starting point with the prescribed Indivo Record ID. When the PHA redirects the IFRAME to the authorization page, Indivo notices that this record has already authorized the app, and simply redirects the IFRAME immediately to the PHA’s callback_url. Thus, a complete OAuth process is re-performed, and the PHA re-obtains an access token, access secret, Indivo record ID and privacy group.

The PHA should never assume that the access token and secret stay the same. The long-term identifier that the PHA should key its data against is the Indivo Record ID.

**Admin Applications**

Admin Applications contact the Indivo X server using 2-legged OAuth only, with just a consumer key and consumer secret.

**Chrome Applications**

Most Indivo developers who only wish to write PHAs can safely ignore Chrome applications. Developers who wish to customize the entire Indivo experience need to understand Chrome apps.

The Indivo Chrome (User Interface) contacts the Indivo X server first using 2-legged OAuth to create a user-specific session using the user’s username and password. Indivo X responds with a fresh OAuth token and secret valid for the length of a typical web session. Then all Indivo Chrome calls to the Indivo X server on behalf of a given user are made as 3-legged calls, using the Indivo Chrome’s consumer key and secret, and the specific session token and secret.

**5.5.2 In-Browser Connect Authentication**

Connect-style authentication enables user applications running framed within the Indivo UI to make API calls solely using javascript, without having to navigate the OAuth dance. Connect-style authentication works as follows:

- When the Indivo UI app opens a user app within its iframe, it acquires a set of OAuth credentials that allows the UI app to make proxied API calls on behalf of the user app, using the API call POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials.
• The Indivo UI app additionally opens a channel to the iframe (using something like jschannel), so that the app can make client-side requests directly to the UI app.

• When the app wants to make an API call, it makes an unsigned request (in the client-side javascript) through its channel.

• The UI app receives the request, signs it with the Connect credentials, and passes the request through to the backend Indivo Server.

• Indivo Server processes the request, and sends the results back to the UI app, which in turn passes them through the channel to the user app.

Thus, the user app needs to perform no authentication when making API calls—it merely passes the calls through its channel to the UI app (a process which we’ve already implemented in our javascript client) and receives the results. The security of the call is enforced in the channel, and by the tokens used by the UI Server to authenticate the call.

5.5.3 Pre-generated REST Authentication

Pre-generated REST Authentication enables user applications running framed within the Indivo UI to acquire oAuth access tokens that can be used sign 3-legged API calls (as with standard Indivo oAuth) without having to navigate the oAuth dance. The authentication process is as follows:

• When the Indivo UI app opens a user app within its iframe, it acquires and preauthorizes an access token for the app, using the API call POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials.

• The UI app then appends a well-formed oAuth header containing the access token to the user app’s start url. The format of the header (and required parameters) are described in the SMART documentation.

• The user app extracts the access token and indivo record ID from the oauth header, and uses it to sign subsequent API calls.

5.6 indivo

5.6.1 indivo package

Subpackages

indivo.data_models package

Subpackages

indivo.data_models.contrib package

Module contents  Contributed Indivo DataModels

indivo.data_models.core package
Module contents  Core Indivo Data Models

class indivo.data_models.core.Procedure(*args, **kwargs)
  Bases: indivo.models.fact.Fact

  Procedure(id, created_at, document_id, record_id, fact_ptr_id, date_performed, name, name_type, name_value, name_abbrev, provider_name, provider_institution, location, comments)

  exception DoesNotExist
    Bases: indivo.models.fact.DoesNotExist

  exception Procedure.MultipleObjectsReturned
    Bases: indivo.models.fact.MultipleObjectsReturned

Procedure._base_manager = <django.db.models.manager.Manager object at 0x2f8ac50>

Procedure._default_manager = <django.db.models.manager.Manager object at 0x2f8ac50>

Procedure._meta = <Options for Procedure>

Procedure.fact_ptr

Procedure.filter_fields = {'name': ('name', 'string'), 'created_at': ('created_at', 'date'), 'comments': ('comments', 'string'), ...

Procedure.objects = <django.db.models.manager.Manager object at 0x2f8ac50>

@classmethod

LabResult.to_rdf(*args, **kwargs)

Exception DoesNotExist
  Bases: indivo.models.fact.DoesNotExist

Exception LabResult.MultipleObjectsReturned
  Bases: indivo.models.fact.MultipleObjectsReturned

LabResult._base_manager = <django.db.models.manager.Manager object at 0x2f92810>

LabResult._default_manager = <django.db.models.manager.Manager object at 0x2f92810>

LabResult._meta = <Options for LabResult>

LabResult.fact_ptr

LabResult.filter_fields = {'narrative_result': ('narrative_result', 'string'), 'collected_by_name_given': ('collected_by_name_given', 'string'), ...

LabResult.objects = <django.db.models.manager.Manager object at 0x2f92810>

LabResult.to_rdf(*args, **kwargs)

class indivo.data_models.core.SimpleClinicalNote(*args, **kwargs)
  Bases: indivo.models.fact.Fact
SimpleClinicalNote(id, created_at, document_id, record_id, fact_ptr_id, date_of_visit, finalized_at, visit_type, visit_type_type, visit_type_value, visit_type_abbrev, visit_location, specialty, specialty_type, specialty_value, specialty_abbrev, signed_at, provider_name, provider_institution, chief_complaint, content)

Exception

DoesNotExist
Bases: indivo.models.fact.DoesNotExist

MultipleObjectsReturned
Bases: indivo.models.fact.MultipleObjectsReturned

SimpleClinicalNote._base_manager = <django.db.models.manager.Manager object at 0x30624d0>
SimpleClinicalNote._default_manager = <django.db.models.manager.Manager object at 0x30624d0>
SimpleClinicalNote._meta = <Options for SimpleClinicalNote>

SimpleClinicalNote.filter_fields = {'specialty_value': ('specialty_value', 'string'), 'finalized_at': ('finalized_at', 'date'), ...

SimpleClinicalNote.get_next_by_date_of_visit(*moreargs, **morekwargs)
SimpleClinicalNote.get_previous_by_date_of_visit(*moreargs, **morekwargs)

SimpleClinicalNote.objects = <django.db.models.manager.Manager object at 0x30624d0>

Immunization(id, created_at, document_id, record_id, fact_ptr_id, date, product_class_identifier, product_class_system, product_class_title, product_class_2_identifier, product_class_2_system, product_class_2_title, refusal_reason_identifier, refusal_reason_system, refusal_reason_title, administration_status_identifier, administration_status_system, administration_status_title, product_name_identifier, product_name_system, product_name_title)

Exception

DoesNotExist
Bases: indivo.models.fact.DoesNotExist

MultipleObjectsReturned
Bases: indivo.models.fact.MultipleObjectsReturned

Immunization._base_manager = <django.db.models.manager.Manager object at 0x3064710>
Immunization._default_manager = <django.db.models.manager.Manager object at 0x3064710>
Immunization._meta = <Options for Immunization>

Immunization.filter_fields = {'product_class_title': ('product_class_title', 'string'), 'administration_status_system': ...

Immunization.objects = <django.db.models.manager.Manager object at 0x3064710>

Immunization.to_rdf(*args, **kwargs)

Equipment(id, created_at, document_id, record_id, fact_ptr_id, date_started, date_stopped, name, vendor, description)

Exception

DoesNotExist
Bases: Indivo.models.fact.DoesNotExist

MultipleObjectsReturned
Bases: Indivo.models.fact.MultipleObjectsReturned

5.6. indivo

437
class indivo.data_models.core.Measurement(*args, **kwargs)
    Bases: indivo.models.fact.Fact
    Measurement(id, created_at, document_id, record_id, fact_ptr_id, type, value, unit, datetime)

    exception DoesNotExist
        Bases: indivo.models.fact.DoesNotExist

    exception Measurement.MultipleObjectsReturned
        Bases: indivo.models.fact.MultipleObjectsReturned

Measurement._base_manager = <django.db.models.manager.Manager object at 0x3060910>
Measurement._default_manager = <django.db.models.manager.Manager object at 0x3060910>
Measurement._meta = <Options for Measurement>
Measurement.fact_ptr
Measurement.filter_fields = {'created_at': ('created_at', 'date'), 'type': ('type', 'string'), 'unit': ('unit', 'string')}
Measurement.get_next_by_datetime(*moreargs, **morekwargs)
Measurement.get_previous_by_datetime(*moreargs, **morekwargs)
Measurement.objects = <django.db.models.manager.Manager object at 0x3060910>
Measurement.set_source_docs(docs)

class indivo.data_models.core.Fill(*args, **kwargs)
    Bases: indivo.models.fact.Fact
    Fill(id, created_at, document_id, record_id, fact_ptr_id, date, dispenseDaysSupply, pbm, medication_id, pharmacy_ncpdpid, pharmacy_adr_region, pharmacy_adr_country, pharmacy_adr_postalcode, pharmacy_adr_city, pharmacy_adr_street, pharmacy_org, provider_tel_1_type, provider_tel_1_preferred_p, provider_tel_1_number, provider_bday, provider_email, provider_npi_number, provider_adr_region, provider_adr_country, provider_adr_postalcode, provider_adr_city, provider_adr_street, provider_gender, provider_tel_2_type, provider_tel_2_preferred_p, provider_tel_2_number, provider_race, provider_dea_number, provider_preferred_language, provider_ethnicity, provider_name_given, provider_name_prefix, provider_name_suffix, provider_name_family, provider_name_middle, quantityDispensed_value, quantityDispensed_unit)

    exception DoesNotExist
        Bases: indivo.models.fact.DoesNotExist

    exception Fill.MultipleObjectsReturned
        Bases: indivo.models.fact.MultipleObjectsReturned

Fill._base_manager = <django.db.models.manager.Manager object at 0x3060910>
Fill._default_manager = <django.db.models.manager.Manager object at 0x3060910>
Fill._meta = <Options for Fill>
Fill.fact_ptr
fill.filter_fields = {'provider_npi_number': ('provider_npi_number', 'string'), 'provider_adr_country': ('provider_adr_country', 'string'),
                   'quantityDispensed_unit': ('quantityDispensed_unit', 'string'), 'provider_adr_city': ('provider_adr_city', 'string')}

fill.get_provider_gender_display(*moreargs, **morekwargs)
fill.get_provider_tel_1_type_display(*moreargs, **morekwargs)
fill.get_provider_tel_2_type_display(*moreargs, **morekwargs)
fill.medication
fill.objects = <django.db.models.manager.Manager object at 0x306fc90>
classmethod fill.to_rdf(*args, **kwargs)

class indivo.data_models.core.Medication(*args, **kwargs)
    Bases: indivo.models.fact.Fact
    Medication(id, created_at, document_id, record_id, fact_ptr_id, endDate, instructions, startDate, prove-
    nance_identifier, provenance_system, provenance_title, frequency_value, frequency_unit, drugName_identifier,
    drugName_system, drugName_title, quantity_value, quantity_unit)

exception DoesNotExist
    Bases: indivo.models.fact.DoesNotExis

exception Medication.MultipleObjectsReturned
    Bases: indivo.models.fact.MultipleObjectsReturned

Medication._base_manager = <django.db.models.manager.Manager object at 0x3069f50>
Medication._default_manager = <django.db.models.manager.Manager object at 0x3069f50>
Medication._meta = <Options for Medication>
Medication.fact_ptr
Medication.filter_fields = {'startDate': ('startDate', 'date'), 'drugName_system': ('drugName_system', 'string')}
Medication.fulfillments
Medication.objects = <django.db.models.manager.Manager object at 0x3069f50>
classmethod Medication.to_rdf(*args, **kwargs)

class indivo.data_models.core.Encounter(*args, **kwargs)
    Bases: indivo.models.fact.Fact
    Encounter(id, created_at, document_id, record_id, fact_ptr_id, startDate, endDate, facil-
    ity_adr_region, facility_adr_country, facility_adr_postalcode, facility_adr_city, facility_adr_street, facility-
    name, encounterType_identifier, encounterType_system, encounterType_title, provider_tel_1_type, provider_tel_1_preferred_p, provider_tel_1_number, provider_bday, provider_email, provider_npi_number, provider_adr_region, provider_adr_country, provider_adr_postalcode, provider_adr_city, provider_adr_street, provider_gender, provider_tel_2_type, provider_tel_2_preferred_p, provider_tel_2_number, provider_race, provider_dea_number, provider_preferred_language, provider_ethnicity, provider_name_given, provider_name_prefix, provider_name_suffix, provider_name_family, provider_name_middle)

exception DoesNotExist
    Bases: indivo.models.fact.DoesNotExis

exception Encounter.MultipleObjectsReturned
    Bases: indivo.models.fact.MultipleObjectsReturned

Encounter._base_manager = <django.db.models.manager.Manager object at 0x3069f50>
Encounter._default_manager = <django.db.models.manager.Manager object at 0x3069f50>
Encounter._meta = <Options for Encounter>
Encounter.fact_ptr

Encounter.filter_fields = {'startDate': ('startDate', 'date'), 'endDate': ('endDate', 'date'), 'provider_adr_country': ('provider_adr_country', 'string'), ...}

Encounter.get_provider_gender_display(*moreargs, **morekwargs)

Encounter.get_provider_tel_1_type_display(*moreargs, **morekwargs)

Encounter.get_provider_tel_2_type_display(*moreargs, **morekwargs)

Encounter.objects = <django.db.models.manager.Manager object at 0x3082710>

Encounter.to_rdf(*args, **kwargs)

Encounter.vitalsigns_set

class indivo.data_models.core.VitalSigns(*args, **kwargs)

Bases: indivo.models.fact.Fact

VitalSigns(id, created_at, document_id, record_id, fact_ptr_id, date, encounter_id, temperature_unit, temperature_value, temperature_name_identifier, temperature_name_system, temperature_name_title, weight_unit, weight_value, weight_name_identifier, weight_name_system, weight_name_title, oxygen_saturation_unit, oxygen_saturation_value, oxygen_saturation_name_identifier, oxygen_saturation_name_system, oxygen_saturation_name_title, bmi_unit, bmi_value, bmi_name_identifier, bmi_name_system, bmi_name_title, respiratory_rate_unit, respiratory_rate_value, respiratory_rate_name_identifier, respiratory_rate_name_system, respiratory_rate_name_title, heart_rate_unit, heart_rate_value, heart_rate_name_identifier, heart_rate_name_system, heart_rate_name_title, bp_diastolic_unit, bp_diastolic_value, bp_diastolic_name_identifier, bp_diastolic_name_system, bp_diastolic_name_title, bp_systolic_unit, bp_systolic_value, bp_systolic_name_identifier, bp_systolic_name_system, bp_systolic_name_title, bp_diastolic_name_identifier, bp_diastolic_name_system, bp_diastolic_name_title, bp_systolic_name_identifier, bp_systolic_name_system, bp_systolic_name_title, bp_method_identifier, bp_method_system, bp_method_title, bp_site_identifier, bp_site_system, bp_site_title, bp_systolic_unit, bp_systolic_value, bp_systolic_name_identifier, bp_systolic_name_system, bp_systolic_name_title, bp_position_identifier, bp_position_system, bp_position_title)

exception DoesNotExist

Bases: indivo.models.fact.DoesNotExist

exception VitalSigns.MultipleObjectsReturned

Bases: indivo.models.fact.MultipleObjectsReturned

VitalSigns._base_manager = <django.db.models.manager.Manager object at 0x307cd10>

VitalSigns._default_manager = <django.db.models.manager.Manager object at 0x307cd10>

VitalSigns._meta = <Options for VitalSigns>

VitalSigns.encounter

VitalSigns.fact_ptr

VitalSigns.filter_fields = {'weight_name_title': ('weight_name_title', 'string'), 'oxygen_saturation_name_identifier': ('oxygen_saturation_name_identifier', 'string'), 'oxygen_saturation_name_system': ('oxygen_saturation_name_system', 'string'), 'oxygen_saturation_name_title': ('oxygen_saturation_name_title', 'string')}

VitalSigns.objects = <django.db.models.manager.Manager object at 0x307cd10>

class indivo.data_models.core.Problem(*args, **kwargs)

Bases: indivo.models.fact.Fact

Problem(id, created_at, document_id, record_id, fact_ptr_id, startDate, endDate, notes, name_identifier, name_system, name_title)

exception DoesNotExist

Bases: indivo.models.fact.DoesNotExist
exception Problem.MultipleObjectsReturned
        Bases: indivo.models.fact.MultipleObjectsReturned

Problem._base_manager = <django.db.models.manager.Manager object at 0x308a7d0>
Problem._default_manager = <django.db.models.manager.Manager object at 0x308a7d0>
Problem._meta = <Options for Problem>

Problem.fact_ptr

Problem.filter_fields = {'startDate': ('startDate', 'date'), 'name_identifier': ('name_identifier', 'string'), 'name_system': ('name_system', 'string'), 'name_title': ('name_title', 'string'), 'endDate': ('endDate', 'date'), 'notes': ('notes', 'string')}

Problem.objects = <django.db.models.manager.Manager object at 0x308a7d0>

classmethod Problem.to_rdf(*args, **kwargs)

class indivo.data_models.core.Allergy(*args, **kwargs)
        Bases: indivo.models.fact.Fact

Allergy(id, created_at, document_id, record_id, fact_ptr_id, category_identifier, category_system, category_title, food_allergen_identifier, food_allergen_system, food_allergen_title, severity_identifier, severity_system, severity_title, drug_allergen_identifier, drug_allergen_system, drug_allergen_title, drug_class_allergen_identifier, drug_class_allergen_system, drug_class_allergen_title, allergic_reaction_identifier, allergic_reaction_system, allergic_reaction_title)

exception DoesNotExist
        Bases: indivo.models.fact.DoesNotExist

exception Allergy.MultipleObjectsReturned
        Bases: indivo.models.fact.MultipleObjectsReturned

Allergy._base_manager = <django.db.models.manager.Manager object at 0x308eb10>
Allergy._default_manager = <django.db.models.manager.Manager object at 0x308eb10>
Allergy._meta = <Options for Allergy>

Allergy.fact_ptr

Allergy.filter_fields = {'drug_class_allergen_system': ('drug_class_allergen_system', 'string'), 'food_allergen_system': ('food_allergen_system', 'string'), 'allergic_reaction_title': ('allergic_reaction_title', 'string')}

Allergy.objects = <django.db.models.manager.Manager object at 0x308eb10>

classmethod Allergy.to_rdf(*args, **kwargs)

class indivo.data_models.core.AllergyExclusion(*args, **kwargs)
        Bases: indivo.models.fact.Fact

AllergyExclusion(id, created_at, document_id, record_id, fact_ptr_id, name_identifier, name_system, name_title)

exception DoesNotExist
        Bases: indivo.models.fact.DoesNotExist

exception AllergyExclusion.MultipleObjectsReturned
        Bases: indivo.models.fact.MultipleObjectsReturned

AllergyExclusion._base_manager = <django.db.models.manager.Manager object at 0x308ef50>
AllergyExclusion._default_manager = <django.db.models.manager.Manager object at 0x308ef50>
AllergyExclusion._meta = <Options for AllergyExclusion>

AllergyExclusion.fact_ptr

AllergyExclusion.filter_fields = {'name_title': ('name_title', 'string'), 'created_at': ('created_at', 'date'), 'name_system': ('name_system', 'string'), 'name_identifier': ('name_identifier', 'string')}
Indivo X Documentation, Release 2.0.0

```
AllergyExclusion.objects = <django.db.models.manager.Manager object at 0x308ef50>
classmethod AllergyExclusion.to_rdf(*args, **kwargs)
```

Submodules

**indivo.data_models.options module**  Options processing and loading for Indivo medical data models.

**class** indivo.data_models.options.DataModelOptions

Bases: object

Defines optional extra functionality for Indivo datamodels.

To add options to a datamodel, subclass this class and override its attributes.

Currently available options are:

- **model_class_name**: Required. The name of the datamodel class to attach to.
- **serializers**: Custom serializers for the data model. Should be set to a subclass of `indivo.serializers.DataModelSerializers`.
- **field_validators**: Custom validators for fields on the data model. A dictionary, where keys are field names on the model, and values are lists of Django Validators to be run against the field.

**classmethod attach**(data_model_class)

Apply these options to a data model class.

Attaches custom serializers and field validators.

**classmethod attach_p**(data_model_class)

True if these options should be applied to data_model_class.

Right now, does `cls.model_class_name` match `data_model_class.__name__`?

**classmethod attach_serializers**(data_model_class)

**classmethod attach_validators**(data_model_class)

**field_validators** = {}

**model_class_name** = ''

**serializers** = None

**Module contents**  Indivo DataModels

**class** indivo.data_models.IndivoDataModelLoader**(top)**

Bases: object

**_discover_python_data_models**(dirpath, fileroot, ext)

Imports a python module and extracts all Indivo Fact subclasses.

**_discover_smdl_data_models**(dirpath, fileroot, ext)

Reads in an SDML model definition and generates Indivo Fact subclasses.

**classmethod add_model_to_module**(model_name, model_class, module)

**classmethod detect_model_dir**(dir_path)

Detects whether a directory is a properly-formatted datamodel.

This is true if:

- It contains a model file of an appropriate type (for now, .py or .smdl)
•More to come later (maybe)

`dir_path` MUST be an absolute path for this to work. Returns a tuple of `(valid_p, fileroot, ext)`, where `valid_p` is True if the format is valid, `fileroot` is the name of the file containing the model definition (without the extension), and `ext` is the extension. If no such file exists, returns `(False, None, None)`. Returns the first valid definition format.

```python
def discover_data_models():
    A generator for iterating over all valid datamodels below `toplevel_dir`.

    At each step, returns a tuple of `(class_name, class)`, where class is a subclass of `indivo.models.Fact` corresponding to a datamodel.

    If a model.py file fails to produce such a class, this function will silently skip it.
```

```python
def import_data_models(target_module):
```

```python
def process_data_model_extras(dirpath, model_class):
    Processes extra options included in an extra.py file for a data model.

    Looks for `indivo.data_models.DataModelOptions` classes and attaches them to the data model
```

```python
def attach_filter_fields(cls):
    `indivo.data_models.load_data_models(from_dir, target_module)`
    Load all datamodels under `directory` into `module`.
```

**indivo.fields package**

**Submodules**

**indivo.fields.dummy_fields module**

```python
class indivo.fields.dummy_fields.AddressField(verbosename=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class 'django.db.models.fields.NOT_PROVIDED' at 0x2778c18>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None):
    Bases: indivo.fields.dummy_fields.DummyField
```

A field for representing a physical address.

Creating an AddressField named ‘address’, for example, will (under the hood) create the fields:

•`address_country`, the country in which the address is located
•`address_city`, the city in which the address is located
•`address_postalcode`, the postalcode of the address
•`address_region`, the region (state, in the US) in which the address is located
•`address_street`, the street address (including street number, apartment number, etc.) at which the address is located
When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original address field name.

\[
\text{db_type}(*\text{args}, **\text{kwargs})
\]

\[
\text{get\_db\_prep\_lookup}(*\text{args}, **\text{kwargs})
\]

\[
\text{get\_db\_prep\_save}(*\text{args}, **\text{kwargs})
\]

\[
\text{get\_db\_prep\_value}(*\text{args}, **\text{kwargs})
\]

\[
\text{replacements} = {'_region': (<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True}), '_country': (<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True}), '_street': (<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True})
\]

class indivo.fields.dummy_fields.BloodPressureField(verbosename=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class 'django.db.models.fields.NOT_PROVIDED at 0x2778c18'>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None)

Bases: indivo.fields.dummy_fields.DummyField

A field for representing a blood pressure measurement.

Creating a BloodPressureField named ‘bp’, for example, will (under the hood) create the fields:

- \text{bp\_position}, the position in which the measurement was taken (a CodedValueField)
- \text{bp\_site}, the site on the body at which the measurement was taken (a CodedValueField)
- \text{bp\_method}, the method of the measurement (a CodedValueField)
- \text{bp\_diastolic}, the diastolic blood pressure (a VitalSignField)
- \text{bp\_systolic}, the systolic blood pressure (a VitalSignField)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original bp field name.

\[
\text{db\_type}(*\text{args}, **\text{kwargs})
\]

\[
\text{get\_db\_prep\_lookup}(*\text{args}, **\text{kwargs})
\]

\[
\text{get\_db\_prep\_save}(*\text{args}, **\text{kwargs})
\]

\[
\text{get\_db\_prep\_value}(*\text{args}, **\text{kwargs})
\]

\[
\text{replacements} = {'_diastolic': (<class 'indivo.fields.dummy_fields.VitalSignField'>, {}), '_method': (<class 'indivo.fields.dummy_fields.CodedValueField'>, {}), '_position': (<class 'indivo.fields.dummy_fields.CodedValueField'>, {}), '_country': (<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True})
\]
class indivo.fields.dummy_fields.CodedValueField (verbose_name=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class django.db.models.fields.NOT_PROVIDED at 0x2778c18>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None)

Bases: indivo.fields.dummy_fields.DummyField

A field for representing coded data elements.

Creating a CodedValueField named ‘value’, for example, will (under the hood) create the fields:

- value_identifier, the system-specific identifier that represents the element (i.e. an RXNorm CUI)
- value_title, the human-readable title of the element
- value_system, the coding system used to represent the element

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original value field name.

db_type (*args, **kwags)

get_db_prep_lookup (*args, **kwags)

get_db_prep_save (*args, **kwags)

get_db_prep_value (*args, **kwags)

replacements = {'_identifier': (<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True}), '_system': (<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True}), '_title': (<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True})}

class indivo.fields.dummy_fields.DummyField (verbose_name=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class django.db.models.fields.NOT_PROVIDED at 0x2778c18>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None)

Bases: django.db.models.fields.Field

A field that should be replaced by other fields.

replacements should be a mapping from field_suffix to (fieldclass, field_kwags). This instructs the datamodel loader to remove this field, and for each entry in the mapping, to add a new field with the original name concatenated with field_suffix, which is an instance of class fieldclass instantiated with field_kwags.

Eventually, when Django supports fields mapping to multiple database columns, these fields should actually manage multiple DB columns, but for now we’re just using strict substitution.
class indivo.fields.dummy_fields.NameField

A field for representing a person’s name.

Creating a NameField named 'name', for example, will (under the hood) create the fields:

• name_family, the family (last) name of the person
• name_given, the given (first) name of the person
• name_middle, the middle name of the person
• name_prefix, the prefix (i.e., 'Mr.', 'Sir', etc.) for the person’s name
• name_suffix, the suffix (i.e., 'Jr.', 'Ph.D.', etc.) for the person’s name

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original name field name.
class indivo.fields.dummy_fields.OrganizationField(verbosename=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class django.db.models.fields.NOT_PROVIDED at 0x2778e18>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None)

Bases: indivo.fields.dummy_fields.DummyField

A field for representing an organization.

Creating an OrganizationField named ‘organization’, for example, will (under the hood) create two fields:

• pharmacy_name, the name of the organization
• organization_adr, the address at which the organization is located (an AddressField)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original organization field name.

db_type(*args, **kwargs)

get_db_prep_lookup(*args, **kwargs)

get_db_prep_save(*args, **kwargs)

get_db_prep_value(*args, **kwargs)

replacements = {'_adr': (<class 'indivo.fields.dummy_fields.AddressField'>, {}), '_name': (<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True})}

class indivo.fields.dummy_fields.PharmacyField(verbosename=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class django.db.models.fields.NOT_PROVIDED at 0x2778c18>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None)

Bases: indivo.fields.dummy_fields.DummyField

A field for representing a pharmacy.

Creating a PharmacyField named ‘pharmacy’, for example, will (under the hood) create three fields:

• pharmacy_ncpdpid, the pharmacy’s National Council for Prescription Drug Programs (NCPDP) ID number
• pharmacy_adr, the address at which the pharmacy is located (an AddressField)
*pharmacy_org*, the name of the organization that owns the pharmacy

When describing instances of your model (either when defining a *transform output* or when referencing fields using the *Indivo Query API*), you must refer to these field names, not the original *pharmacy* field name.

```python
db_type(*args, **kwargs)
get_db_prep_lookup(*args, **kwargs)
get_db_prep_save(*args, **kwargs)
get_db_prep_value(*args, **kwargs)
```

```text
replacements = {'_ncpdpid': (class 'django.db.models.fields.CharField', {'max_length': 255, 'null': True}), '_adr': (class 'indivo.fields.address.AddressField', {}), '_org': (class 'django.db.models.fields.CharField', {'max_length': 255, 'null': True})}
```

```python
class indivo.fields.dummy_fields.ProviderField(verbose_name=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class 'django.db.models.fields.NOT_PROVIDED at 0x2778c18'>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None)
```

Bases: indivo.fields.dummy_fields.DummyField

A field for representing a medical provider.

Creating a ProviderField named ‘doc’, for example, will (under the hood) create the fields:

- *doc_dea_number*, the provider’s Drug Enforcement Agency (DEA) number
- *doc_ethnicity*, the provider’s ethnicity
- *doc_npi_number*, the provider’s National Provider Identification (NPI) number
- *doc_preferred_language*, the provider’s preferred language
- *doc_race*, the provider’s race
- *doc_adr*, the provider’s address (an AddressField)
- *doc_bday*, the provider’s birth date
- *doc_email*, the provider’s email address
- *doc_name*, the provider’s name (a NameField)
- *doc_tel_1*, the provider’s primary phone number (a TelephoneField)
- *doc_tel_2*, the provider’s secondary phone number (a TelephoneField)
- *doc_gender*, the provider’s gender, limited to *m* (male) or *f* (female)

When describing instances of your model (either when defining a *transform output* or when referencing fields using the *Indivo Query API*), you must refer to these field names, not the original *doc* field name.

```python
db_type(*args, **kwargs)
gender_choices = (('m', 'male'), ('f', 'female'))
get_db_prep_lookup(*args, **kwargs)
get_db_prep_save(*args, **kwargs)
```
```python

def get_db_prep_value(*args, **kwargs)
    replacements = {'_tel_1': (<class 'indivo.fields.dummy_fields.TelephoneField'>, {'max_length': 255, 'null': True}), 'bday': (<class 'indivo.fields.dummy_fields.BirthDateField'>, {'max_length': 255, 'null': True}), '_name': (<class 'indivo.fields.dummy_fields.NameField'>, {'max_length': 255, 'null': True})}

class indivo.fields.dummy_fields.QuantitativeResultField (verbose_name=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class 'django.db.models.fields.NOT_PROVIDED'>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None)

    Bases: indivo.fields.dummy_fields.DummyField

    A field for representing a quantitative result, and expected ranges for that result.

    Creating a QuantitativeResultField named ‘lab_result’, for example, will (under the hood) create the fields:

- **lab_result_non_critical_range**, the range outside of which results are ‘critical’ (a ValueRangeField)
- **lab_result_normal_range**, the range outside of which results are ‘abnormal’ (a ValueRangeField)
- **lab_result_value**, the actual result (a ValueAndUnitField)

    When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original lab_result field name.

  db_type(*args, **kwargs)
  get_db_prep_lookup(*args, **kwargs)
  get_db_prep_save(*args, **kwargs)
  get_db_prep_value(*args, **kwargs)
  replacements = {'_value': (<class 'indivo.fields.dummy_fields.ValueAndUnitField'>, {}), '_non_critical_range': (<class 'indivo.fields.dummy_fields.ValueRangeField'>, {}), '_normal_range': (<class 'indivo.fields.dummy_fields.ValueRangeField'>, {})}
```
class `indivo.fields.dummy_fields.TelphoneField`:

```python
(verbosename=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class django.db.models.fields.NOT_PROVIDED at 0x2778c18>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None)
```

Bases: `indivo.fields.dummy_fields.DummyField`

A field for representing a telephone number.

Creating a `TelephoneField` named `phone`, for example, will (under the hood) create the fields:

- `phone_type`, The type of the phone number, limited to `h` (home), `w` (work), or `c` (cell)
- `phone_number`, The actual phone number
- `phone_preferred`, Whether or not this number is a preferred method of contact (True or False)

When describing instances of your model (either when defining a `transform output` or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `phone` field name.

```python
db_type(*args, **kwargs)
get_db_prep_lookup(*args, **kwargs)
get_db_prep_save(*args, **kwargs)
get_db_prep_value(*args, **kwargs)
```

```python
phone_number_type_choices = (('h', 'Home'), ('w', 'Work'), ('c', 'Cell'))
```

```python
replacements = {'_type': (<class 'django.db.models.fields.CharField'>, {'max_length': 1, 'null': True, 'choices': (('h', 'Home'), ... {'default': False}), '_number': (<class 'django.db.models.fields.CharField'>, {'max_length': 20, 'null': True})}
```

A field for representing data elements with both a value and a unit.

Creating a `ValueAndUnitField` named `frequency`, for example, will (under the hood) create the fields:

```python
(verbosename=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class django.db.models.fields.NOT_PROVIDED at 0x2778c18>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None)
```

Bases: `indivo.fields.dummy_fields.DummyField`

A field for representing data elements with both a value and a unit.

Creating a `ValueAndUnitField` named `frequency`, for example, will (under the hood) create the fields:
• `frequency_value`, the value of the element
• `frequency_unit`, the units in which the value is measured

When describing instances of your model (either when defining a `transform output` or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `frequency` field name.

```python
db_type(*args, **kwargs)
get_db_prep_lookup(*args, **kwargs)
get_db_prep_save(*args, **kwargs)
get_db_prep_value(*args, **kwargs)
replacements = {'_value':(<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True}), '_unit':(<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True})}
```

class `indivo.fields.dummy_fields.ValueRangeField`(
    verbose_name=None,
    name=None,
    primary_key=False,
    max_length=None,
    unique=False,
    blank=False,
    null=False,
    db_index=False,
    rel=None,
    default=<class 'django.db.models.fields.NOT_PROVIDED at 0x2778c18'>,
    editable=True,
    serialize=True,
    unique_for_date=None,
    unique_for_month=None,
    unique_for_year=None,
    choices=None,
    help_text='',
    db_column=None,
    db_tablespace=None,
    auto_created=False,
    validators=[],
    error_messages=None)
```

Bases: `indivo.fields.dummy_fields.DummyField`

A field for representing a range of values.

Creating a `ValueRangeField` named ‘normal_range’, for example, will (under the hood) create the fields:

• `normal_range_max`, the maximum value of the range (a `ValueAndUnitField`)
• `normal_range_min`, the minimum value of the range (a `ValueAndUnitField`)

When describing instances of your model (either when defining a `transform output` or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `normal_range` field name.

```python
db_type(*args, **kwargs)
get_db_prep_lookup(*args, **kwargs)
get_db_prep_save(*args, **kwargs)
get_db_prep_value(*args, **kwargs)
replacements = {'_min':(<class 'indivo.fields.dummy_fields.ValueAndUnitField'>, {}), '_max':(<class 'indivo.fields.dummy_fields.ValueAndUnitField'>, {})}
```
### class indivo.fields.dummy_fields.VitalSignField

```python
class indivo.fields.dummy_fields.VitalSignField(verbosename=None, name=None, primary_key=False, max_length=None, unique=False, blank=False, null=False, db_index=False, rel=None, default=<class django.db.models.fields.NOT_PROVIDED at 0x2778c18>, editable=True, serialize=True, unique_for_date=None, unique_for_month=None, unique_for_year=None, choices=None, help_text='', db_column=None, db_tablespace=None, auto_created=False, validators=[], error_messages=None)
```

**Bases:** indivo.fields.dummy_fields.DummyField

A field for representing a single measurement of a vital sign.

Creating a VitalSignField named ‘bp’, for example, will (under the hood) create the fields:

- **bp_unit**, the unit of the measurement
- **bp_value**, the value of the measurement
- **bp_name**, the name of the measurement (a CodedValueField)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original bp field name.

- **db_type** (*args, **kwargs)
- **get_db_prep_lookup** (*args, **kwargs)
- **get_db_prep_save** (*args, **kwargs)
- **get_db_prep_value** (*args, **kwargs)

replacements = {'_unit': (<class 'django.db.models.fields.CharField'>, {'max_length': 255, 'null': True}), '_value': (<class 'django.db.models.fields.FloatField'>, {'null': True}), '_name': (<class 'indivo.fields.dummy_fields.CodedValueField'>, {})}

---

**Module contents**  
Indivo Fields.

Custom-defined Django Model Field subclasses used for representing Medical Data via the Django ORM.

- **indivo.serializers package**

**Submodules**

- **indivo.serializers.base module**  
Module for abstract serializer base classes.

Differentiated from the Django base serializer by allowing implementing serializers to process nested objects. Recursion is prevented by keeping track of visited objects, and results in repeated objects being skipped after their first appearance

```python
class indivo.serializers.base.DeserializedObject (obj, m2m_data=None)
```

**Bases:** object

Deserialization is not currently supported

```python
save (save_m2m=True)
```
class indivo.serializers.baseDeserializer

Bases: object

Deserialization is not supported

exception indivo.serializers.base.SerializationRecursionError

Bases: exceptions.Exception

Object encountered twice during serialization

class indivo.serializers.base.Serializer

Bases: django.core.serializers.base.Serializer

Abstract serializer base class.

handle_o2m_field

Called to handle a OneToMany field.

internal_use_only = False

process_object

process_objects

serialize

Serialize a queryset.

indivo.serializers.datamodel_hooks module

class indivo.serializers.datamodel_hooks.DataModelSerializers

Bases: object

Abstract base class for defining serializers that should be attached to a data model class.

Serializers will override the default implementations. Subclasses should define any of three methods:

• to_rdf(queryset, result_count, record=None, carenet=None): takes a queryset, and formats it as valid RDF/XML string.
• to_xml(queryset, result_count, record=None, carenet=None): takes a queryset, and formats it as a valid XML string.
• to_json(queryset, result_count, record=None, carenet=None): takes a queryset, and formats it as a valid JSON string.

In order to be called, the methods must be attached to that data model class by calling the attach_to_data_model() method.

classmethod attach_to_data_model(data_model_cls)

Add all of the defined methods as classmethods on data_model_cls.

indivo.serializers.json module Indivo JSON Serializer

class indivo.serializers.json.IndivoJSONEncoder(skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, encoding='utf-8', default=None)

Bases: json.encoder.JSONEncoder

Encodes datetime/date/time as ISO8601

default (o)
class indivo.serializers.json.Serializer
    Bases: indivo.serializers.python.Serializer
    
    Convert a queryset to JSON.
    
    end_serialization()
    getvalue()

    internal_use_only = False

indivo.serializers.python module   Indivo Python Serializer

indivo.serializers.pythonDeserializer(object_list, **options)

    Deserialization is not currently supported

class indivo.serializers.python.Serializer
    Bases: indivo.serializers.base.Serializer
    
    Serializes a QuerySet to basic Python objects. - Parses foreign key and many-to-many fields into nested objects
    - Output mirrors Indivo SDML format

    end_object(obj)
    end_serialization()
    getvalue()

    handle_field(obj, field)
    handle_fk_field(obj, field)
    handle_m2m_field(obj, field)
    handle_o2m_field(obj, field_name)

    internal_use_only = True

    start_object(obj)
    start_serialization()

indivo.serializers.xml_serializer module   Indivo custom XML serializer

class indivo.serializers.xml_serializer.Deserializer(stream_or_string, **options)
    Bases: indivo.serializers.base.Deserializer

    Deserialization is not currently supported

class indivo.serializers.xml_serializer.Serializer
    Bases: indivo.serializers.base.Serializer
    
    Serializes a QuerySet to XML. Foreign Key and Many-to-Many fields are processed as sub-objects.

    end_object(obj)
        Called after handling all fields for an object.

    end_serialization()
        End serialization – end the document.

    getvalue()

    handle_field(obj, field)
        Called to handle each field on an object (except for ForeignKeys and ManyToManyFields)
handle_fk_field \((obj, field)\)
Called to handle a ForeignKey (we need to treat them slightly differently from regular fields).

handle_m2m_field \((obj, field)\)
Called to handle a ManyToManyField.

handle_o2m_field \((obj, field_name)\)

start_object \((obj)\)
Called as each object is handled.

start_serialization ()
Start serialization

Module contents  Indivo serializers

indivo.views package

Subpackages

indivo.views.documents package

Submodules

indivo.views.documents.document module

indivo.views.documents.document.__local_document_create \((request, record, pha, external_id, existing_doc)\)
Create a document, or update one in place.
This function serves document_create() and document_create_or_update(), which encompasses record- and/or app-specific documents.
The external_id is expected to be already adjusted.

Arguments:
• request: The incoming Django HttpRequest object.
• record: if the document is record-specific, this Record instance refers to the document’s record.
• pha: if the document is application-specific, this PHA instance refers to the application to which the document pertains.
• external_id: the external identifier to assing to the new document, if available. The identifier should already have been prepared using prepare_external_id().
• existing_doc: If the new document will overwrite (via in-place update or versioning) an existing document, this Document instance references the old document to be overwritten.

Returns:
• An HttpResponse object whose body is a string of XML describing the created document, ready for return over the wire on success.
• An instance of django.http.HttpResponseBadRequest if the new document failed validation during the creation process.

Raises:
django.db.IntegrityError: if the arguments to this function violate a database unique constraint (i.e., duplicate external id).

Warning: If an IntegrityError is raised, it will invalidate the current database transaction. Calling functions should handle this case and rollback the current transaction.

indivo.views.documents.document._document_create(creator, content, pha, record, replaces_document=None, external_id=None, mime_type=None, status=None)

Create an Indivo Document.

This is the lowest-level creation function called for all record- and/or app-specific documents.

The PHA argument, if non-null, indicates app-specificity only. By this point, the external_id should be fully formed.

If status is specified, then it is used, otherwise it is not specified and the model’s default value is used (i.e. active).

This function creates a new model instance, processing the document if necessary, and storing it in the database (or in the file system, if the document is binary).

Arguments:

• creator: The Principal instance that is responsible for creating the document.
• content: The raw content (XML or binary) of the document to be created.
• pha: if the document is application-specific, this PHA instance refers to the application to which the document pertains.
• record: if the document is record-specific, this Record instance refers to the document’s record.
• replaces_document: If the new document will overwrite (via in-place update or versioning) an existing document, this Document instance references the old document to be overwritten.
• external_id: the external identifier to assing to the new document, if available. The identifier should already have been prepared using prepare_external_id().
• mime_type: the mime type of the new document, i.e. application/xml.
• status: The initial status of the new document. active by default.

Returns:

• A new instance of Document, on success. If the document was updated in place, and no new document was created, the old document is returned.

Raises:

• ValueError: if the document doesn’t validate.
• django.db.IntegrityError: if the arguments to this function violate a database unique constraint (i.e., duplicate external id).

Warning: If an IntegrityError is raised, it will invalidate the current database transaction. Calling functions should handle this case and rollback the current transaction.

indivo.views.documents.document._get_document(record=None, carenet=None, document_id=None, pha=None, external_id=None)

Fetch a document from the DB.
EITHER `document_id` OR `external_id` must be provided (exclusive or). If a document doesn’t exist matching all passed arguments, this call returns None.

Arguments:

- **record**: if the document is record-specific, this `Record` instance refers to the document’s record.
- **carenet**: if the document is being found via a carenet, this `Carenet` instance refers to the carenet containing the document.

**Warning**: Carenet membership is NOT checked in this function. That security must be checked elsewhere.

- **document_id**: the internal identifier for the document, if available.

**Note**: One of `external_id` or `document_id` MUST be passed to this function, or it cannot retrieve a unique document.

- **pha**: if the document is application-specific, this `PHA` instance refers to the application to which the document pertains.
- **external_id**: the external identifier for the document, if available. The identifier should already have been prepared using `prepare_external_id()`.

**Note**: One of `external_id` or `document_id` MUST be passed to this function, or it cannot retrieve a unique document.

Returns:

- An instance of `Document`, on success.
- None, if a document satisfying all passed arguments could not be found.

```python
indivo.views.documents.document._render_document(document)
```

Get the raw content of a document, ready to be sent over the wire.

Arguments:

- **document**: the `Document` instance to render.

Returns:

- An `HttpResponse` object whose body contains the The raw content of the `Document` on success.

```python
indivo.views.documents.document._render_documents(docs, record, pha, tdc, format_type='xml')
```

Lowlevel document rendering to response data.

Arguments:

- **docs**: An iterable of `Document` objects to be rendered.
- **record**: The `Record` that every document in `docs` belongs to.
- **pha**: The `PHA` that `docs` are scoped to.
- **tdc**: The total document count of objects to render (i.e., `len(docs)`). This can be passed in to avoid recomputing the size of `docs` if that might be expensive (i.e., a QuerySet which would require an extra DB call).
- **format_type**: The format to render into. Options are `xml`.

Returns:
• an HTTPResponse whose body is an XML string containing the rendered list of documents (which might be empty).

indivo.views.documents.document._set_doc_latest(doc)
Set the ‘latest version’-related fields on a document for rendering.
This is computed at load-time before each rendering of document. It is not pre-computed or stored in the DB.
returns: None

indivo.views.documents.document.app_document_create(*args, **kwargs)
Create an app-specific Indivo document.
Calls into document_create_or_update().

indivo.views.documents.document.app_document_create_or_update(*args, **kwargs)
Create or Overwrite an app-specific Indivo document.
Calls into document_create_or_update().

indivo.views.documents.document.app_document_create_or_update_ext(*args, **kwargs)
Create an app-specific Indivo document with an associated external id.
Calls into document_create_or_update().

indivo.views.documents.document.app_document_list(request, *args, **kwargs)
List app-specific documents.
Calls into document_list().

indivo.views.documents.document.app_specific_document(request, pha, document_id)
Retrieve an app-specific document.
Calls into document().

indivo.views.documents.document.document(request, document_id, record=None, pha=None)
Retrieve a document, record- and/or app-specific.

Arguments:
• request: the incoming Django HttpRequest object.
• document_id: the internal identifier of the document to retrieve.
• record: if the document is record-specific, this Record instance refers to the document’s record.
• pha: if the document is application-specific, this PHA instance refers to the application to which the document pertains.

Returns:
• An HttpResponse object whose body contains the raw content of the document on success.

Raises:
• A django.http.Http404 if the document could not be found.

indivo.views.documents.document.document_create(*args, **kwargs)
Create a record-specific Indivo Document.
Calls into __local_document_create().

indivo.views.documents.document.document_create_by_ext_id(*args, **kwargs)
Create a record-specific Indivo Document with an associated external id.
Calls into __local_document_create().
indivo.views.documents.document.document_create_or_update(request, pha, record=None, document_id=None, external_id=None)

Create or Overwrite an app-specific or record-app-specific document, possibly with an associated external id.
Prepares the external id, loads the existing document to overwrite, then calls into __local_document_create().

indivo.views.documents.document.document_list(request, query_options, record=None, pha=None)

List Indivo documents.

Arguments:

• **request**: The incoming Django HttpRequest object. request.GET may contain:
  – **type**: The Indivo document schema type on which to filter the result set. As of 2010-08-16, type is no longer part of the URL, it’s only in the GET query parameters.
  – **limit, offset, status, order_by**: Standard paging and filtering arguments. See marsloader() or API Query Interface.
  – **record**: if desired documents are record-specific, this Record instance refers to the record to filter on.
  – **pha**: if the desired documents are application-specific, this PHA instance refers to the app to filter on.

Returns:

• an HTTPResponse whose body is an XML string containing the rendered list of documents (which might be empty), on success.

Raises:

• **django.http.Http404**: if type was passed, but didn’t correspond to an existing Indivo schema.

indivo.views.documents.document.record_app_document_create(*args, **kwargs)

Create a record-app-specific Indivo document.
Calls into document_create_or_update().

indivo.views.documents.document.record_app_document_create_or_update_ext(*args, **kwargs)

Create or Overwrite a record-app-specific Indivo document with an associated external id.
Calls into document_create_or_update().

indivo.views.documents.document.record_app_document_list(request, *args, **kwargs)

List record-app-specific documents.
Calls into document_list().

indivo.views.documents.document.record_app_specific_document(request, record, pha, document_id)

Retrieve a record-app-specific document.
Calls into document().

indivo.views.documents.document.record_document_list(request, *args, **kwargs)

List record-specific documents.
Calls into document_list().

indivo.views.documents.document.record_specific_document(request, record, document_id)

Retrieve a record-specific document.
Calls into `document()`.

**indivo.views.documents.document_delete module**

`indivo.views.documents.document_delete._document_delete(document_id, pha=None, record=None)`

Delete a document.

**ARGUMENTS:**

- `document_id`: The internal identifier of the document to delete.
- `pha`: If the document to delete is scoped to an app, this PHA instance refers to the app.
- `record`: If the document to delete is scoped to a record, this Record instance refers to the record.

**RETURNS:**

- 200 OK on success.

**RAISES:**

- `django.http.Http404` if the arguments don’t identify an existing document.

`indivo.views.documents.document_delete.app_document_delete(request, pha, document_id)`

Delete an app-specific document.

No restrictions, since this storage is managed by the app.

Calls into `_document_delete()`.

`indivo.views.documents.document_delete.documents_delete(request, record)`

Delete all documents associated with a record.

**ARGUMENTS:**

- `request`: The incoming Django HttpRequest object.
- `record`: The Record to purge of documents.

**RETURNS:**

- 200 OK on success.

`indivo.views.documents.document_delete.record_app_document_delete(request, record, pha, document_id)`

Delete a record-app-specific document.

No restrictions, since this storage is managed by the app.

Calls into `_document_delete()`.

**indivo.views.documents.document_label module**

`indivo.views.documents.document_label._document_label(request, document_id=None, external_id=None, pha=None, app_specific=False)`

Set a document’s label.

**ARGUMENTS:**

- `request`: The incoming Django HttpRequest object. `request.POST` must consist of a raw string containing the new label to assign.
• **record**: The Record that the document is scoped to, if applicable.

• **document_id**: The internal identifier of the document to re-label.

**Note**: One of `external_id` or `document_id` MUST be passed to this function, or it cannot retrieve a unique document.

• **external_id**: The external identifier of the document to re-label.

**Note**: One of `external_id` or `document_id` MUST be passed to this function, or it cannot retrieve a unique document.

• **pha**: The PHA object that the document is scoped to. Also serves to scope `external_id`, if present and `app_specific` is `True`.

• **app_specific**: Whether or not the document is app-specific. The mere presence of the `pha` argument isn’t enough to satisfy this question, as `pha` might have been passed in only to scope an external id for a non-app-specific document.

**RETURNS:**

• An HttpResponse object with an XML string describing the re-labeled document on success.

**RAISES:**


```python
indivo.views.documents.document_label.app_document_label(request, pha, document_id)
```

Set the label of an app-specific document.

Calls into `_document_label()`.

```python
indivo.views.documents.document_label.record_app_document_label(request, record, pha, document_id)
```

Set the label of a record-app-specific document.

Calls into `_document_label()`.

```python
indivo.views.documents.document_label.record_document_label(request, record, document_id)
```

Set the label of a record-specific document.

Calls into `_document_label()`.

```python
indivo.views.documents.document_label.record_document_label_ext(request, record, document_id=None, external_id=None, pha=None, app_specific=False)
```

Set the label of a record-specific document, specified by external id.

Calls into `_document_label()`.

```python
indivo.views.documents.document_meta module
```

5.6. indivo
indivo.views.documents.document_meta._document_meta

Fetch the metadata of a single document.

Metadata includes:

- id
- date created
- creator
- the document that replaced this one
- the document that this one replaces
- the original document in the version chain
- the latest document in the version chain
- label
- current status
- nevershare status
- related documents

ARGUMENTS:

- record: The Record that the document is scoped to, if applicable.
- carenet: The Carenet that the document is shared into, if applicable.
- document: The document to get metadata for, if it has been prefetched.
- pha: The PHA object that the document is scoped to. Also serves to scope external_id, if present and app_specific is True.
- external_id: The external identifier of the document to re-label.

Note: One of external_id or document MUST be passed to this function, or it cannot retrieve a unique document.

• app_specific: Whether or not the document is app-specific. The mere presence of the pha argument isn’t enough to satisfy this question, as pha might have been passed in only to scope an external id for a non-app-specific document.

RETURNS:

• An HttpResponse object with an XML string describing the document metadata on success.

RAISES:

• django.http.Http404 if document isn’t passed and external_id doesn’t identify an existing document.
indivo.views.documents.document_meta.app_document_meta_ext(request, pha, external_id)

Fetch the metadata of an app-specific document identified by external id.
Calls into _document_meta().

indivo.views.documents.document_meta.carenet_document_meta(request, carenet, document_id)

Fetch the metadata of a record-specific document via a carenet.
Calls into _document_meta().

indivo.views.documents.document_meta.record_app_document_meta(request, pha, document_id)

Fetch the metadata of a record-app-specific document.
Calls into _document_meta().

indivo.views.documents.document_meta.record_app_document_meta_ext(request, pha, external_id)

Fetch the metadata of a record-app-specific document identified by external id.
Calls into _document_meta().

indivo.views.documents.document_meta.record_document_meta(request, record, document_id)

Fetch the metadata of a record-specific document.
Calls into _document_meta().

indivo.views.documents.document_meta.record_document_meta_ext(request, record, pha, external_id)

Fetch the metadata of a record-specific document identified by external id.
Calls into _document_meta().

indivo.views.documents.document_meta.update_document_meta(request, record, document_id)

Set metadata fields on a document. NOT IMPLEMENTED.

indivo.views.documents.document_rels module

indivo.views.documents.document_rels._document_create_by_rel(request, record, document_id, rel, pha=None, external_id=None)

Create a document and relate it to an existing document.

ARGUMENTS:

• request: The incoming Django HttpRequest object. request.POST must contain the raw content of the new document.
• record: The Record to which to scope the new document, and to which the source document is scoped.
• document_id: The internal document identifier for the source document.
• rel: The relationship type to establish between the source document and the new document (as a string).
• pha: The PHA object that scopes the external_id, if present.
• external_id: The external identifier to assign to the newly created document.
RETURNS:

- An HttpResponse object whose body is a string of XML describing the created document, ready for return over the wire on success.
- 400 Bad Request if the new document content is invalid

RAISES:

- Django.http.Http404 if document_id doesn’t identify an existing document scoped to record, or if rel doesn’t identify a valid relationship type.

indivo.views.documents.document_rels.document_create_by_rel(*args, **kwargs)
Create a document and relate it to an existing document.
Calls into _document_create_by_rel().

indivo.views.documents.document_rels.document_create_by_rel_with_ext_id(*args, **kwargs)
Create a document, assign it an external id, and relate it to an existing document.
Calls into _document_create_by_rel().

indivo.views.documents.document_rels.document_rels(request, record, document_id_0, rel, document_id_1)
Create a new relationship between two existing documents.

ARGUMENTS:

- request: The incoming Django HttpRequest object.
- record: The Record that the documents are scoped to.
- document_id_0: The internal document identifier for the source document.
- rel: The relationship type between the two documents (as a string).
- document_id_1: The internal document identifier for the related document.

RETURNS:

- 200 OK on success.

RAISES:

- Django.http.Http404 if either document_id_0 or document_id_1 don’t identify an existing document scoped to record, or if rel doesn’t identify a valid relationship type.

indivo.views.documents.document_rels.get_documents_by_rel(request, *args, **kwargs)
Get all documents related to the passed document_id by a relation of the passed relation-type.
Includes relationships to other versions of document_id. Paging operators are NOT IMPLEMENTED.

ARGUMENTS:

- request: The incoming Django HttpRequest object.
- record: The Record that the document is scoped to.
- document_id: The internal document identifier for the source document.
- rel: The relationship type to filter related documents by (as a string).
- limit, offset, status, order_by: Standard paging and filtering arguments. See marsloader() or API Query Interface.

Note: Paging operators are not implemented for this call currently. Passing them into the function will have no effect on output.
•*pha*: The *PHA* object that the source document is scoped to, if applicable.

**RETURNS:**
•An HttpResponse object with an XML string listing related documents on success.

**RAISES:**
•django.http.Http404 if *document_id* doesn’t identify an existing document scoped to *record*.

indivo.views.documents.document_status module

indivo.views.documents.document_status.document_set_status(*args, **kwargs)

Set the status of a record-specific document.

**ARGUMENTS:**
•*request*: The incoming Django HttpRequest object. *request.POST* must contain:
  –*status*: The new status for the document. Must identify an existing StatusName object.
  –*reason*: The reason for the status change
•*record*: The Record that the document is scoped to.
•*document_id*: The internal identifier of the document whose status is being altered.

**RETURNS:**
•200 OK on success.
•400 Bad Request if *request.POST* is missing arguments.

**RAISES:**
•django.http.Http404 if *document_id* doesn’t identify an existing document scoped to *record*.

indivo.views.documents.document_status.document_status_history(*request, record, document_id*)

List all changes to a document’s status over time.

**ARGUMENTS:**
•*request*: The incoming Django HttpRequest object.
•*record*: The Record that the document is scoped to.
•*document_id*: The internal identifier of the document for which to get status history.

**RETURNS:**

**RAISES:**
•django.http.Http404 if *document_id* doesn’t identify an existing document scoped to *record*.

indivo.views.documents.document_versions module

indivo.views.documents.document_versions._document_version(*request, record, document_id, pha=None, external_id=None*)

Create a new version of a record-specific document.

**ARGUMENTS:**
**request**: The incoming Django HttpRequest object. `request.POST` must consist of a raw string containing the new document content.

**record**: The Record to which the old document is scoped, and to which the new document will be scoped.

**document_id**: The internal identifier of the old document. The old document must be at the latest version, or the call will fail.

**external_id**: The external identifier to assign to the new document.

**pha**: The PHA object used to scope `external_id`, if present.

**RETURNS:**

- An HttpResponse object with an XML string containing metadata on the new document on success.
- 400 Bad Request if the old document has previously been replaced by a newer version.

**RAISES:**

- `django.http.Http404` if `document_id` doesn’t identify an existing document, or if document creation fails (odd behavior).

```python
django.views.documents.document_versions.document_version(*args, **kwargs)
```

Create a new version of a record-specific document.

Calls into `_document_version()`.

```python
django.views.documents.document_versions.document_version_by_ext_id(*args, **kwargs)
```

Create a new version of a record-specific document and assign it an external id.

Calls into `_document_version()`.

```python
django.views.documents.document_versions.document_versions(request, *args, **kwargs)
```

Retrieve the versions of a document.

**ARGUMENTS:**

- `request`: The incoming Django HttpRequest object.
- `record`: The Record to which the document is scoped.
- `document_id`: The internal identifier of the document.
- `limit`, `offset`, `status`, `order_by`: Standard paging and filtering arguments. See `marsloader()` or API Query Interface.

**RETURNS:**

- An HttpResponse object with an XML string containing metadata on all versions of the document, including the passed `document_id`, on success.

**RAISES:**


```python
django.views.documents.special_documents._read_demographics(request, record_or_carenet)
```

Read demographics from a record or carenet.

Calls into `get_demographics()`.
indivo.views.documents.special_documents.get_demographics(record_or_carenet)
Fetch the demographics from either a record or a carenet.

ARGUMENTS:
• record_or_carenet: The Record or Carenet from which to fetch the demographics.

RETURNS:
• The demographics as a Demographics instance, if they exist.
• None, if record or carenet hasn’t been assigned demographics

RAISES:
• ValueError if record_or_carenet is None.

indivo.views.documents.special_documents.read_demographics(request, record)
Read demographics from a record.
Calls into _read_demographics().

indivo.views.documents.special_documents.read_demographics_carenet(request, carenet)
Read demographics from a carenet.
Calls into _read_demographics().

indivo.views.documents.special_documents.set_demographics(*args, **kwargs)
Create or update demographics on a record.

ARGUMENTS:
• request: The incoming Django HttpRequest object. request.POST must consist of a raw string containing the demographics content.
• record: The Record from which to fetch the demographics.

RETURNS:
• a django.http.HttpResponse containing Metadata XML on the newly created document. TODO: what should we return now that we have a model
• 400 Bad Request if the new demographics content didn’t validate.

Module contents Indivo Document Views.
The views that implement document-related API calls.

indivo.views.reports package

Submodules

indivo.views.reports.ccr module
indivo.views.reports.ccr.report_ccr(request, record=None, carenet=None)
Export patient data as a Continuity of Care Record (CCR) document.
Will return 200 OK with a CCR on success, 400 Bad Request if neither a record or carenet was passed.
**Indivo X Documentation, Release 2.0.0**

**indivo.views.reports.equipment module**

`indivo.views.reports.equipment._equipment_list(request, *args, **kwargs)`  
List the equipment objects matching the passed query parameters.

See *API Query Interface* for a listing of valid parameters.

Will return 200 OK with a list of equipment on success, 400 Bad Request if any invalid query parameters were passed.

`indivo.views.reports.equipment.carenet_equipment_list(*args, **kwargs)`  
List the equipment data for a given carenet.

For 1:1 mapping of URLs to views. Just calls `_equipment_list()`.

`indivo.views.reports.equipment.equipment_list(*args, **kwargs)`  
List the equipment data for a given record.

For 1:1 mapping of URLs to views. Just calls `_equipment_list()`.

**indivo.views.reports.generic module**

`indivo.views.reports.generic._generic_list(request, query_options, data_model, record=None, carenet=None, response_format=None)`  
List the Model objects matching the passed query parameters.

See *API Query Interface* for a listing of valid parameters.

Will return 200 OK with a list of Models or AggregateReports on success, 400 Bad Request if any invalid query parameters were passed.

`indivo.views.reports.generic.aggregate_json(query)`  
Serialize an aggregate query’s results to a JSON string

`indivo.views.reports.generic.aggregate_xml(query)`  
Serialize an aggregate query’s results to an XML string

`indivo.views.reports.generic.carenet_generic_list(request, *args, **kwargs)`  
List the Model data for a given carenet.

`indivo.views.reports.generic.generic_list(request, *args, **kwargs)`  
List the Model data for a given record.

`indivo.views.reports.generic.serialize(cls, format, query, record=None, carenet=None)`  
Serialize an indivo.lib.query to the requested format

Non-aggregate queries are handled by the data model’s own serialization methods, while aggregate queries are serialized in a standard way to AggregateReports

**Returns:**

- A string representation of the serialized query results in the requested format

`indivo.views.reports.generic.serialize_as_aggregate(format, query)`  
Serialize an aggregate query to the requested format

**indivo.views.reports.measurement module**

`indivo.views.reports.measurement._measurement_list(request, *args, **kwargs)`  
List the measurement objects matching the passed query parameters.

See *API Query Interface* for a listing of valid parameters.

Will return 200 OK with a list of measurements on success, 400 Bad Request if any invalid query parameters were passed.
indivo.views.reports.measurement.carennet_measurement_list(*args, **kwargs)

List the measurement data for a given carenet.

For 1:1 mapping of URLs to views. Just calls _measurement_list().

indivo.views.reports.measurement.measurement_list(*args, **kwargs)

List the measurement data for a given record.

For 1:1 mapping of URLs to views. Just calls _measurement_list().

indivo.views.reports.procedure module

indivo.views.reports.procedure._procedure_list(request, *args, **kwargs)

List the procedure objects matching the passed query parameters.

See API Query Interface for a listing of valid parameters.

Will return 200 OK with a list of procedures on success, 400 Bad Request if any invalid query parameters were passed.

indivo.views.reports.procedure.carennet_procedure_list(*args, **kwargs)

List the procedure data for a given carenet.

For 1:1 mapping of URLs to views. Just calls _procedure_list().

indivo.views.reports.procedure.procedure_list(*args, **kwargs)

List the procedure data for a given record.

For 1:1 mapping of URLs to views. Just calls _procedure_list().

indivo.views.reports.simple_clinical_notes module

indivo.views.reports.simple_clinical_notes._simple_clinical_notes_list(request, *args, **kwargs)

List the simple_clinical_notes objects matching the passed query parameters.

See API Query Interface for a listing of valid parameters.

Will return 200 OK with a list of simple clinical notes on success, 400 Bad Request if any invalid query parameters were passed.

indivo.views.reports.simple_clinical_notes.carennet_simple_clinical_notes_list(*args, **kwargs)

List the simple_clinical_notes data for a given carenet.

For 1:1 mapping of URLs to views. Just calls _simple_clinical_notes_list().

indivo.views.reports.simple_clinical_notes.simple_clinical_notes_list(*args, **kwargs)

List the simple_clinical_notes data for a given record.

For 1:1 mapping of URLs to views. Just calls _simple_clinical_notes_list().

indivo.views.reports.smart module

indivo.views.reports.smart.get_default_query_args()

indivo.views.reports.smart.smart_allergies(request, record)

SMART allergy list, serialized as RDF/XML.

A bit more complicated than the generic list view, since we have to serialize AllergyExclusions as well.

indivo.views.reports.smart.smart_allergies_instance(request, record, model_id)

Retrieve a specific instance of a SMART allergy.

SMART allergies can be an Allergy or an AllergyExclusion
Indivo X Documentation, Release 2.0.0

indivo.views.reports.smart.smart_generic(request, record, model_name)
SMART-compatible alias for the generic list view: returns data_models serialized as SMART RDF.

indivo.views.reports.smart.smart_generic_instance(request, record, model_name, model_id)
Retrieve a specific instance of a SMART model.

Module contents  Indivo Reporting Views.
The views that implement reporting calls.

indivo.views.shares package

Submodules

indivo.views.shares.shares_account module
indivo.views.shares.shares_account.account_permissions(request, account)
List the carenets that an account has access to.

Will return 200 OK with a list of carenets on success.

indivo.views.shares.shares_account.carennet_account_create(request, carenet)
Add an account to a carenet.

request.POST must contain:

• account_id: The email of the account to share with.

• write: Whether or not the account can write to the carenet. Can be ‘true’ or ‘false’. This is currently unused, as carenets are read-only, however it must be provided anyways.

Will return 200 OK on success, 404 Not Found if the specified account_id doesn’t exist, or 400 Bad Request if either account_id or write is not in request.POST.

indivo.views.shares.shares_account.carennet_account_delete(request, account, carenet)
Remove an account from a carenet.

Will return 200 OK on success.

indivo.views.shares.shares_account.carennet_account_list(request, carenet)
List the accounts in a carenet.

Will return 200 OK with a list of accounts on success.

indivo.views.shares.shares_account.carennet_account_permissions(request, carenet, account)
List the permissions of an account within a carenet.

Currently, carenets are read-only and accounts can access all documents within a carenet, so this call returns static XML indicating blanket access.

Will return 200 OK with the static XML on success.

indivo.views.shares.shares_autoshare module
indivo.views.shares.shares_autoshare.autoshare_create(request, record, carenet)
Automatically share all documents of a certain type into a carenet.

request.POST must contain:
• **type**: the document schema namespace to create an autoshare for

Will return **200 OK** on success, **404 Not Found** if the specified **type** doesn’t exist.

```python
indivo.views.shares.shares_autoshare.autoshare_delete(request, record, carenet)
```

Remove an autoshare from a carenet.

`request.POST` must contain:

• **type**: the document schema namespace to remove an autoshare for

This will effectively unshare all documents of type **type** from the carenet, except documents which were shared individually.

Will return **200 OK** on success, **404 Not Found** if the specified **type** doesn’t exist.

```python
indivo.views.shares.shares_autoshare.autoshare_list(request, record)
```

For a single record, list all carenets that a given doctype is autoshared with.

`request.GET` must contain:

• **type**: the document schema namespace to check autoshares for

Will return **200 OK** with a list of carenets that have an autoshare set up for doctype **type** on success, **404 Not Found** if the specified **type** doesn’t exist.

```python
indivo.views.shares.shares_autoshare.autoshare_list_bytype_all(request, record)
```

For a single record, list all doctypes autoshared into carenets.

Will return **200 OK** with a list of doctypes and the carenets that have an autoshare for each doctype on success.

```python
indivo.views.shares.shares_autoshare.autoshare_revert(request, record, document_id, carenet)
```

Revert the document-sharing of a document in a carenet to whatever rules are specified by autoshares. **NOT IMPLEMENTED**.

```python
indivo.views.shares.shares_autoshare.carenets_nevershare(request, document_id)
```

### `indivo.views.shares.shares_carenets module`

```python
indivo.views.shares.shares_carenets.carenet_create(*args, **kwargs)
```

Create a new carenet for a record.

`request.POST` must contain:

• **name**: the label for the new carenet.

Will return **200 OK** with XML describing the new carenet on success, **400 Bad Request** if **name** wasn’t passed or if a carenet named **name** already exists on this record.

```python
indivo.views.shares.shares_carenets.carenet_delete(request, carenet)
```

Delete a carenet.

Will return **200 OK** on success.

```python
indivo.views.shares.shares_carenets.carenet_list(request, record)
```

List all carenets for a record.

Will return **200 OK** with a list of carenets on success.

```python
indivo.views.shares.shares_carenets.carenet_rename(*args, **kwargs)
```

Change a carenet’s name.

`request.POST` must contain:

• **name**: The new name for the carenet.
Will return 200 OK with XML describing the renamed carenet on success, 400 Bad Request if name wasn’t passed or if a carenet named name already exists on this record.

indivo.views.shares.shares_document module

indivo.views.shares.shares_document.carenets (request, record, document_id)

List all the carenets into which a document has been shared.

Will return 200 OK with a list of carenets on success, 404 Not Found if document_id is invalid.

indivo.views.shares.shares_document module

indivo.views.shares.shares_document.carenets (request, record, document_id)

List all the carenets into which a document has been shared.

Will return 200 OK with a list of carenets on success, 404 Not Found if document_id is invalid.

indivo.views.shares.shares_nevershare module

472 Chapter 5. Indivo Hackers
Remove the nevershare flag from a document.

If a document has was shared via autoshare or explicitly, then marked as nevershare, this call will reactivate all previously existing shares.

Will return 200 OK on success, 404 Not Found if document_id is invalid.

Flag a document to never be shared, anywhere.

This overrides autoshares and existing shares, and prevents sharing the document in the future, until document_remove_nevershare() is called.

Will return 200 OK on success, 404 Not Found if document_id is invalid.

Retrieve the permissions for an app within a carenet. NOT IMPLEMENTED.

Add an app to a carenet

Read/write capability is determined by the user who uses the app, not by the app itself, so no permissions are assigned here.

Apps can only be added to carenets if they have already been shared with the carenet’s record (i.e. the user has agreed to use the app).

Autonomous apps cannot be added to carenets, as they require a full-record scope.

Will return 200 OK on success, 404 Not Found if there is no existing share between pha and carenet’s record, or 400 Bad Request if pha is autonomous.

Remove an app from a given carenet.

Will return 200 OK on success, or if pha was never in carenet and no work needed to be done.

List Apps within a given carenet.

Will return 200 OK with manifests for the apps on success.

The views that implement sharing API calls.

Add a new method of authentication to an account.
Accounts cannot be logged into unless there exists a mechanism for authenticating them. Indivo supports one built-in mechanism, password auth, but is extensible with other mechanisms (i.e., LDAP, etc.). If an external mechanism is used, a UI app is responsible for user authentication, and this call merely registers with indivo server the fact that the UI can handle auth. If password auth is used, this call additionally registers the password with indivo server. Thus, this call can be used to add internal or external auth systems.

request.POST must contain:

• **system**: The identifier (a short slug) associated with the desired auth system. password identifies the internal password system, and external auth systems will define their own identifiers.

• **username**: The username that this account will use to authenticate against the new authsystem

• **password**: The password to pair with the username. ONLY REQUIRED IF THE AUTH SYSTEM IS THE INTERNAL PASSWORD SYSTEM.

Will return 200 OK on success, 403 Forbidden if the indicated auth system doesn’t exist, and 400 Bad Request if the POST data didn’t contain a system and a username (and a password if system was password), or if the account is already registered for the given authsystem, or a different account is already registered for the given authsystem with the same username.

indivo.views.account.account_check_secrets(request, account, primary_secret)

Validate an account’s primary and secondary secrets.

If the secondary secret is to be validated, request.GET must contain:

• **secondary_secret**: The account’s secondary secret.

This call will validate the primary secret, and the secondary secret if passed.

Will return 200 OK on success, 403 Forbidden if either validation fails.

indivo.views.account.account_create(*args, **kwargs)

Create a new account, and send out initialization emails.

request.POST holds the creation arguments.

In Demo Mode, this call automatically creates new records for the account, populated with sample data. See Sample Data in Indivo for details.

Required Parameters:

• **account_id**: an identifier for the new address. Must be formatted as an email address.

Optional Parameters:

• **full_name**: The full name to associate with the account. Defaults to the empty string.

• **contact_email**: A valid email at which the account holder can be reached. Defaults to the account_id parameter.

• **primary_secret_p**: 0 or 1. Whether or not to associate a primary secret with the account. Defaults to 1.

• **secondary_secret_p**: 0 or 1. Whether or not to associate a secondary secret with the account. Defaults to 0.

After creating the new account, this call generates secrets for it, and then emails the user (at contact_email) with their activation link, which contains the primary secret.

This call will return 200 OK with info about the new account on success, 400 Bad Request if account_id isn’t provided or isn’t a valid email address, or if an account already exists with an id matching account_id.

indivo.views.account.account_forgot_password(request, account)

Resets an account if the user has forgotten its password.
This is a convenience call which encapsulates account_reset(), account_resend_secret(), and account_secret(). In summary, it resets the account to an uninitialized state, emails the user with a new primary-secret, and returns the secondary secret for display.

Will return 200 OK with the secondary secret on success, 400 Bad Request if the account hasn’t yet been initialized and couldn’t possibly need a reset. If the account has no associated secondary secret, the return XML will be empty.

indivo.views.account.account_info(request, account)
Display information about an account.
Return information includes the account’s secondary-secret, full name, contact email, login counts, state, and auth systems.
Will return 200 OK on success, with account info XML.

indivo.views.account.account_info_set(request, account)
Set basic information about an account.
request.POST can contain any of:
•contact_email: A new contact email for the account.
•full_name: A new full name for the account.
Each passed parameter will be updated for the account.
Will return 200 OK on success, 400 Bad Request if the POST data contains none of the settable parameters.

indivo.views.account.account_initialize(*args, **kwargs)
Initialize an account, activating it.
After validating primary and secondary secrets, changes the account’s state from uninitialized to active and sends a welcome email to the user.
If the account has an associated secondary secret, request.POST must contain:
•secondary_secret: The secondary_secret generated for the account.
Will return 200 OK on success, 403 Forbidden if the account has already been initialized or if either of the account secrets didn’t validate, and 400 Bad Request if a secondary secret was required, but didn’t appear in the POST data.

indivo.views.account.account_password_change(request, account)
Change a account’s password.
request.POST must contain:
•old: The existing account password.
•new: The desired new password.
Will return 200 OK on success, 403 Forbidden if the old password didn’t validate, 400 Bad Request if the POST data didn’t contain both an old password and a new one.

indivo.views.account.account_password_set(request, account)
Force the password of an account to a given value.
This differs from account_password_change() in that it does not require validation of the old password. This function is therefore admin-facing, whereas account_password_change() is user-facing.
request.POST must contain:
•password: The new password to set.
Will return 200 OK on success, 400 Bad Request if the passed POST data didn’t contain a new password.
indivo.views.account.account_primary_secret(request, account)
Display an account’s primary secret.
This is an admin-facing call, and should be used sparingly, as we would like to avoid sending primary-secrets over the wire. If possible, use account_check_secrets() instead.
Will return 200 OK with the primary secret on success.

indivo.views.account.account_resend_secret(request, account)
Sends an account user their primary secret in case they lost it.
Will return 200 OK on success.

indivo.views.account.account_reset(request, account)
Reset an account to an uninitialized state.
Just calls into reset().
Will return 200 OK on success.

indivo.views.account.account_secret(request, account)
Return the secondary secret of an account.
Will always return 200 OK. If the account has no associated secondary secret, the return XML will be empty.

indivo.views.account.account_search(request)
Search for accounts by name or email.
request.GET must contain the query parameters, any of:
• fullname: The full name of the account
• contact_email: The contact email for the account.
This call returns all accounts matching any part of any of the passed query parameters: i.e. it ORs together the query parameters and runs a partial-text match on each.
Will return 200 OK with XML describing matching accounts on success, 400 Bad Request if no query parameters are passed.

indivo.views.account.account_set_state(request, account)
Set the state of an account.
request.POST must contain:
• state: The desired new state of the account.
Options are:
• active: The account is ready for use.
• disabled: The account has been disabled, and cannot be logged into.
• retired: The account has been permanently disabled, and will never allow login again. Retired accounts cannot be set to any other state.
Will return 200 OK on success, 403 Forbidden if the account has been retired and 400 Bad Request if POST data did not contain a “status” parameter

indivo.views.account.account_username_set(request, account)
Force the username of an account to a given value.
request.POST must contain:
• username: The new username to set.
Will return 200 OK on success, 400 Bad Request if the POST data doesn’t contain a new username.
indivo.views.account.delete_user_preferences(request, account, pha)
Delete all app-specific User Preferences for an account.

indivo.views.account.get_connect_credentials(request, account, pha)
Get oAuth credentials for an app to run in Connect or SMART REST mode.
Generates access tokens for pha to run against the record_id specified in request.POST, authorized by account. Generates 2 tokens: one for SMART Connect use, and one for SMART REST use.
If the app is not yet enabled for the record/carenet, this will return a 403 Forbidden.

indivo.views.account.get_user_preferences(request, account, pha)
Get app-specific User Preferences for an account.
We’re just storing these as app-specific documents with a specific external ID. ID is “{account_id}_USER_PREFERENCES”. Note that this will be further prepared by prepare_external_id() before insertion into the database.

indivo.views.account.set_user_preferences(request, account, pha)
Set app-specific User Preferences for an account.
Overrides all existing preferences.

indivo.views.audit module
indivo.views.audit.audit_document_view(request, *args, **kwargs)
Return audits of calls touching record and document_id.
Will return 200 OK with matching audits on success, 404 Not Found if record or document_id don’t exist.
Deprecated since version 0.9.3: Use audit_query() instead.

indivo.views.audit.audit_function_view(request, *args, **kwargs)
Return audits of calls to function_name touching record and document_id.
Will return 200 OK with matching audits on success, 404 Not Found if record or document_id don’t exist.
Deprecated since version 0.9.3: Use audit_query() instead.

indivo.views.audit.audit_query(request, *args, **kwargs)
Select Audit Objects via the Query API Interface.
Accepts any argument specified by the API Query Interface, and filters available audit objects by the arguments.
Will return 200 OK with XML containing individual or aggregated audit records on success, 400 Bad Request if any of the arguments to the query interface are invalid.

indivo.views.audit.audit_record_view(request, *args, **kwargs)
Return audits of calls touching record.
Will return 200 OK with matching audits on success, 404 Not Found if record doesn’t exist.
Deprecated since version 0.9.3: Use audit_query() instead.

indivo.views.base module

indivo.views.messaging module
indivo.views.messaging._get_subject(request)
Extract a message subject from request.POST.

indivo.views.messaging.account_inbox(request, *args, **kwargs)
List messages in an account’s inbox.
Messages will be ordered by order_by and paged by limit and offset. request.GET may additionally contain:
• **include_archive**: Adds messages that have been archived (which are normally omitted) to the listing. Any value will be interpreted as True. Defaults to False, as if it weren’t passed.

Will return 200 OK with a list of messages on success.

**indivo.views.messaging.account_inbox_message** *(request, account, message_id)*

Retrieve an individual message from an account’s inbox.

This call additionally filters message content based on its body-type. For example, markdown content is scrubbed of extraneous HTML, then converted to HTML content. Also, this call marks the message as read.

*message_id* should be the internal identifier of the message

Will return 200 OK with XML describing the message (id, sender, dates received, read, and archived, subject, body, severity, etc.) on success.

**indivo.views.messaging.account_inbox_message_attachment_accept** *(request, account, message_id, attachment_num)*

Accept a message attachment into the record it corresponds to.

This call is triggered when a user views a message with an attachment, and chooses to add the attachment contents into their record.

Will return 200 OK on success, 410 Gone if the attachment has already been saved.

**indivo.views.messaging.account_message_archive** *(request, account, message_id)*

Archive a message.

This call sets a message’s archival date as now, unless it’s already set. This means that future calls to *account_inbox()* will not display this message by default.

Will return 200 OK on success.

**indivo.views.messaging.account_notifications** *(request, *args, **kwargs)*

List an account’s notifications.

Orders by *order_by*, pages by *limit* and *offset*.

Will return 200 OK with a list of notifications on success.

**indivo.views.messaging.account_send_message** *(*args, **kwargs)*

Send a message to an account.

Account messages have no attachments for now, as we wouldn’t know which record to store them on.

request.POST may contain any of:

• **message_id**: An external identifier for the message, used for idempotent sends. Defaults to *None*.

• **subject**: The message subject. Defaults to *[no subject]*.

• **body**: The message body. Defaults to *[no body]*.

• **severity**: The importance of the message. Options are *low*, *medium*, *high*. Defaults to *low*.

After delivering the message to Indivo’s inbox, this call will send an email to the account’s contact address, alerting them that a new message has arrived.

Will return 200 OK on success, 400 Bad Request if the passed *message_id* is a duplicate.

**indivo.views.messaging.record_message_attach** *(*args, **kwargs)*

Attach a document to an Indivo message.

Only XML documents are accepted for now. Since Message objects are duplicated for each recipient account, this call may attach the document to multiple Message objects.
request.POST must contain the raw XML attachment data.

*message_id*: The external identifier of the message to add the attachment to

Will return **200 OK** on success, **400 Bad Request** if the attachment with number *attachment_num* has already been uploaded.

```python
def record_send_message(*args, **kwargs):
    # Send a message to a record.
```

*request.POST* may contain any of:

- **subject**: The message subject. Defaults to [no subject].
- **body**: The message body. Defaults to [no body].
- **body_type**: The formatting of the message body. Options are plaintext, markdown. Defaults to plaintext.
- **num_attachments**: The number of attachments this message requires. Attachments are uploaded with calls to `record_message_attach()`, and the message will not be delivered until all attachments have been uploaded. Defaults to 0.
- **severity**: The importance of the message. Options are low, medium, high. Defaults to low.

*message_id*: An external identifier for the message, used for idempotent sends. Defaults to None.

After delivering the message to the Indivo inbox of all accounts authorized to view messages for the passed *record*, this call will send an email to each account’s contact address, alerting them that a new message has arrived.

Will return **200 OK** on success, **400 Bad Request** if the passed *message_id* is a duplicate.

```python
def _pha(*args, **kwargs):

def _phas(*args, **kwargs):

def all_manifests(request):
    # List SMART manifests for all available userapps.
    Will return **200 OK** with a list of app manifests as JSON on success.

def all_phas(request):
    # List all available userapps.
    Will return **200 OK** with a list of app manifests as JSON on success.

def app_manifest(request, pha):
    # Return a SMART manifest for a single userapp.
    Will return **200 OK** with the app’s JSON manifest on success.

def app_record_list(request, pha):
    # Return a list of all records that have this pha enabled.
    Will return :http:statuscode’200’ with a list of records on success.

def autonomous_access_token(request, pha, record):
    # Fetch an access token for an autonomous app to access a record.
    This call assumes that the app has already been enabled on the record, and that the user has already authorized it (this must be checked in the access control for the function). Otherwise, this will automatically enable the app on the record (a BAD idea).```
This call should be made by autonomous apps to get access tokens for records which have already enabled them (presumably after a call to app_record_list).

Will return 200 OK with a valid access token for the app bound to the record on success.

```
indivo.views.pha.exchange_token(request)
```

Exchange a request token for a valid access token.

This call requires that the request be signed with a valid oauth request token that has previously been authorized.

Will return 200 OK with the access token on success, 403 Forbidden if the oauth signature is missing or invalid.

```
indivo.views.pha(pha, request)
```

Return a description of a single userapp.

Will return 200 OK with the app’s JSON manifest on success.

```
indivo.views.pha.pha_delete(request, pha)
```

Delete a userapp from Indivo.

This call removes the app entirely from indivo, so it will never be accessible again. To remove an app just from a single record, see pha_record_delete().

Will return 200 OK on success.

```
indivo.views.pha.pha_record_delete(request, record, pha)
```

Remove a userapp from a record.

This is accomplished by deleting the app from all carenets belonging to the record, then removing the Shares between the record and the app.

Will return 200 OK on success, 404 Not Found if either the record or the app don’t exist.

```
indivo.views.pha.record_pha_enable(request, record, pha)
```

Enable a userapp for a record.

This is accomplished by adding a share between the record and the app. We don’t limit the share to a carenet: this is scoped to the whole record.

Will return 200 OK on success or if the share already exists, 404 Not Found if either the record or the app doesn’t exist.

```
indivo.views.pha.request_token(request)
```

Get a new request token, bound to a record or carenet if desired.

request.POST may contain EITHER:
- `indivo_record_id`: The record to which to bind the request token.
- `indivo_carenet_id`: The carenet to which to bind the request token.

Will return 200 OK with the request token on success, 403 Forbidden if the oauth signature on the request was missing or faulty.

```
indivo.views.pha.request_token_approve(request, reqtoken)
```

Indicate a user’s consent to bind an app to a record or carenet.

request.POST must contain EITHER:
- `record_id`: The record to bind to.
- `carenet_id`: The carenet to bind to.

Will return 200 OK with a redirect url to the app on success, 403 Forbidden if `record_id/carenet_id` don’t match `reqtoken`. 
**indivo.views.pha.request_token_claim**(request, reqtoken)
Claim a request token on behalf of an account.
After this call, no one but request.principal will be able to approve reqtoken.
Will return 200 OK with the email of the claiming principal on success, 403 Forbidden if the token has already been claimed.

**indivo.views.pha.request_token_info**(request, reqtoken)
Get information about a request token.
Information includes:
- the record/carenet it is bound to
- Whether the bound record/carenet has been authorized before
- Information about the app for which the token was generated.
Will return 200 OK with the info on success.

**indivo.views.pha.session_create**(request)
Authenticate a user and register a web session for them.
request.POST must contain:
- *username*: the username of the user to authenticate.
request.POST may contain EITHER:
  - *password*: the password to use with username against the internal password auth system.
  - *system*: An external auth system to authenticate the user
Will return 200 OK with a valid session token on success, 400 Bad Request if no username was provided, 403 Forbidden if the passed credentials were invalid or the passed system doesn’t exist.

**indivo.views.pha.surl_verify**(request)
Verify a signed URL.
The URL must contain the following GET parameters:
  - *surl_timestamp*: when the url was generated. **Must be within the past hour**, to avoid permitting old surls.
  - *surl_token* The access token used to sign the url.
  - *surl_sig* The computed signature (base-64 encoded sha1) of the url.
Will always return 200 OK. The response body will be one of:
  - `<result>ok</result>`: The surl was valid.
  - `<result>old</result>`: The surl was too old.
  - `<result>mismatch</result>`: The surl’s signature was invalid.

**indivo.views.record module**

**indivo.views.record._record_create**(request, principal_email=None, external_id=None)
Create an Indivo record.
request.POST must contain raw XML that is a valid Indivo Demographics document (see Indivo Document Demographics Schema).
This call will create a new record containing the following information:
  - *creator*: Corresponds to request.principal.
• **label**: The full name of the new record, specified in the demographics document.

• **owner**: Corresponds to `request.principal`.

• **external_id**: An external identifier for the record, if passed in.

Additionally, this call will create a Demographics document for the record.

Will return **200 OK** with information about the record on success, **400 Bad Request** if the demographics data in `request.POST` was empty or invalid XML.

indivo.views.record.*record*(request, record)

Get information about an individual record.

Will return **200 OK** with information about the record on success.

indivo.views.record.*record_create*(**args**, **kwargs**)

Create a new record.

For 1:1 mapping of URLs to views: just calls `_record_create()`.

indivo.views.record.*record_create_ext*(**args**, **kwargs**)

Create a new record with an associated external id.

For 1:1 mapping of URLs to views: just calls `_record_create()`.

indivo.views.record.*record_get_owner*(request, record)

Get the owner of a record.

Will always return **200 OK**. The response body will contain the owner’s email address, or the empty string if the record is unowned.

indivo.views.record.*record_list*(request, **args**, **kwargs**)

List all available records for an account.

This includes records that `account` owns, records that have been fully shared with `account`, and records that are shared with `account` via carenets.

Will return **200 OK** with a list of records on success.

indivo.views.record.*record_notify*(request, record)

Send a notification about a record to all accounts authorized to be notified.

Notifications should be short alerts, as compared to full inbox messages, and may only be formatted as plaintext. `request.POST` must contain:

• **content**: The plaintext content of the notification.

`request.POST` may contain:

• **document_id**: The document to which this notification pertains.

• **app_url**: A callback url to the app for more information.

Will return **200 OK** on success, **400 Bad Request** if `content` wasn’t passed.

indivo.views.record.*record_pha*(**args**, **kwargs**)

Get information about a given userapp bound to a record.

Will return **200 OK** with a JSON manifest for the app on success, **404 Not Found** if the app isn’t actually bound to the record.

indivo.views.record.*record_pha_setup*(**args**, **kwargs**)

Bind an app to a record without user authorization.
This call should be used to set up new records with apps required for this instance of Indivo to run (i.e. syncer apps that connect to data sources). It can only be made by admins, since it skips the normal app authorization process.

request.POST may contain raw content that will be used as a setup document for the record.

Will return **200 OK** with a valid access token for the app bound to the record on success.

```python
indivo.views.record.record_phas(*args, **kwargs)
```

List userapps bound to a given record.

request.GET may optionally contain:

• *type*: An XML schema namespace. If specified, only apps which explicitly declare themselves as supporting that namespace will be returned.

Will return **200 OK** with a list of JSON manifests for the matching apps on success.

```python
indivo.views.record.record_search(request)
```

Search for records by label (usually the same as full name).

request.GET must contain the query parameters, any of:

• *label*: The record’s label

This call returns all records matching any part of any of the query parameters: i.e. it ORs together the query parameters and runs a partial-text match on each.

Will return **200 OK** with XML describing matching records on success, **400 Bad Request** if no query parameters are passed.

```python
indivo.views.record.record_set_owner(request, record)
```

Set the owner of a record.

request.POST must contain the email address of the new owner.

Will return **200 OK** with information about the new owner, **400 Bad Request** if request.POST is empty or the passed email address doesn’t correspond to an existing principal.

```python
indivo.views.record.record_share_add(request, record)
```

Fully share a record with another account.

A full share gives the recipient account full access to all data and apps on the record, and adds the recipient to the list of accounts who are alerted when the record gets a new alert or notification.

request.POST must contain:

• *account_id*: the email address of the recipient account.

request.POST may contain:

• *role_label*: A label for the share (usually the relationship between the record owner and the recipient account, i.e. ‘Guardian’)

Will return **200 OK** on success, **400 Bad Request** if *account_id* was not passed, and **404 Not Found** if the passed *account_id* does not correspond to an existing Account.

```python
indivo.views.record.record_share_delete(request, record, other_account_id)
```

Undo a full record share with an account.

Will return **200 OK** on success, **404 Not Found** if *other_account_id* doesn’t correspond to an existing Account.

```python
indivo.views.record.record_shares(request, record)
```

List the shares of a record.

This includes shares with apps (phashares) and full shares with accounts (fullshares).
Will return 200 OK with a list of shares on success.

**Indivo.views.smart_container module**

- `indivo.views.smart_container.smart_capabilities(request)`
  SMART Capabilities
- `indivo.views.smart_container.smart_ontology(request)`
  Fetch the SMART ontology as RDF/XML.

**Module contents** Indivo Views.

The Django views that implement all Indivo API functionality, including API calls related to:
- accounts
- auditing
- documents
- messaging
- apps
- records
- reporting
- sharing

- `indivo.views._get_indivo_version(smart_version)`
- `indivo.views._get_smart_version(indivo_version)`
- `indivo.views._get_version()`
- `indivo.views.get_version(request)`
  Return the current version of Indivo.

**Submodules**

**indivo.validators module**

Validators useful for attaching to data models.

- `class indivo.validators.ExactValueValidator(valid_value, nullable=False)`
  **Bases**: indivo.validators.ValueInSetValidator
  Validates that a value is exactly equal to a certain value.
  The optional ‘nullable’ flag determines whether or not the value may also be empty.

- `class indivo.validators.NonNullValidator`
  **Bases**: object
  Validates that a value is not null.
  A ‘null’ value is anything that Django would store as NULL in a database: None, '', [], (), or {}. Note that other python objects that evaluate to false (0, False) are not actually null values, as they represent data.
  This validator is useful for validating that strings are non-empty, for example.
class indivo.validators.ValueInSetValidator (valid_values, nullable=False)
    Bases: object
    Validates that a value is within a set of possible values.
    The optional ‘nullable’ flag determines whether or not the value may also be empty.

Module contents

The Indivo Application.
A Django app which implements a Personally-Controlled Health Record platform, exposing an oAuth authenticated, RESTful API for substitutable medical apps.

indivo.check_safety()
    Ensure that the Authorization Module is in place before serving API calls.

5.7 Indivo Basic Data Formats

5.7.1 Timestamps

Using the standard XML Schema for timestamps in the UTC time zone, e.g. 2009-05-04T12:34:56Z.

5.7.2 Null Values

Null elements are represented by removal of the element (deemed optional in the schema if nullable). Null attribute values are represented by the removal of the attribute (also deemed optional in the schema if nullable).

5.8 Indivo Client Libraries

An Indivo client library is any code (in any language) provided as a standard package to app developers which enables them to make API calls against Indivo without worrying about low-level implementation details such as OAuth signing. We currently support a few simple libraries: this document lists those libraries, and provides advice for generating new libraries.

5.8.1 Supported Client Libraries

Currently, we have released the following client libraries:

Python Client

Our python client is a simple wrapper around the commonly used and supported Python Oauth2 library for making OAuth-signed REST calls.
Our code is available on github, and documentation is available in the Indivo docs.
iOS Framework

The Indivo iOS Framework is an object-oriented wrapper that provides class-based access to core Indivo data-types using the Indivo API.

The code is available on github, and documentation is available in the Indivo docs.

SMART Clients

Since Indivo now supports the SMART API, apps written using SMART client libraries will also run on Indivo. See their documentation <http://dev.smartplatforms.org> for more information

Note: Make sure the the SMART clients you use support SMART v0.4

5.8.2 Building a Client Library

A client library’s responsibilities are simple: it must be able to sign HTTP requests using OAuth, send them to Indivo Server or an Indivo UI App (for OAuth authorization), and present the results of the requests back to the app developer. Most languages have libraries for doing these things already, so building a new Indivo client library is actually quite simple.

In order to facilitate auto-generation of clients, Indivo provides an api.xml file, which describes all of the calls a complete client should support. This file can be found in the indivo server source code, in indivo_server/api.xml.

The api.xml file should be updated whenever the supported API calls are modified. In order to insure that you have the most up-to-date version of the file, you can run (from a valid indivo_server checkout):

```
python manage.py generate_api_spec
```

Which will update the api.xml file to be consistent with the current codebase.

5.9 Coding Systems

For Indivo, and in general for a number of health applications, coding systems are used for interoperability. Examples include vaccine disease codes, allergy codes, procedure codes, etc. This page documents a web-based mechanism for documenting and publishing, in a machine-readable manner, these coding systems.

5.9.1 Abstract Model

Beyond the basic attributes (name, publisher, description), a coding system includes:

- a way to list all codes
- a way to look up a single code
- a way to search for codes matching a simple text query (e.g. “diab” should match “diabetes.”)

A single code entry will have, at least:

- a code
- an abbreviation
- a full title
• (optionally) a description
• (optionally) relationships to other codes.

5.9.2 Data Representation

JSON

5.9.3 RESTful Calls

The URL templates define RESTful calls to obtain a single code, and to search for a number of codes. Specifically, given the example above, the following URL returns a single code “123”:

http://codes.indivo.org/systems/allergies/123

And the following URL searches the list of allergy codes for “peanut”:

http://codes.indivo.org/systems/allergies/search?q=peanut

5.9.4 Sources

The coding systems used in Indivo X are as follows. Individual installations need to download the coding systems on their own, as the licenses for these do not permit redistribution, so we cannot package them with Indivo.

Immunizations: HL7 v3

The easiest way to get the HL7 V3 file in vertical-bar-separated format, as required by the coding system loader, is to use bioontology.org.

We specifically used the REST service at http://rest.bioontology.org/bioportal. The ontology code we used to download our version appears to no longer exist, so we’ll look into the latest codes soon. In the meantime, documentation for the REST service is at http://www.bioontology.org/wiki/index.php/NCBO_REST_services

Labs: LOINC

Available at http://loinc.org

There is an encoding issue which forced us to truncate the LOINC file for now at line 43504.

Problems: SNOMED CT


An encoding conversion is required to get to utf8, should be doable using the iconv program on most Linux installations.

Medications: RxTerms


Note that we may move to RxNorm instead of RxTerms.
5.10 API Query Interface

When a list of results are returned, the URL ends in a / and the HTTP method is a GET, as is typical of REST design. In that case, Indivo X supports a generic query string that determines paging and ordering of the results:

?offset={offset}&limit={limit}&order_by={order_by}&status={document_status}

- offset indicates which item number to start with, e.g. when getting a second batch of items.
- limit indicates the maximum number of items to return. This is used in combination with offset to accomplish paging.
- order_by is dependent on the fields returned in the list of items, and each call must thus define which fields are valid. Using an invalid field in order_by results in no effect on the output, as if order_by were absent.
- status can be used where applicable. It pertains to the status of documents and can currently be set to one of three options: ‘void’, ‘archived’ or ‘active’

For minimal (url ends in /reports/minimal/{report_type}/) and generic reports, we expose an expanded query interface, allowing for filtering, date constraints, and aggregation. Note that this interface only makes sense to implement for reports that function solely to retrieve and display fact objects. The interface will not be implemented for any more complex report types that deal with multiple fact objects (what would it mean for a CCR report to group by lab type?). Other reports will still have access to the Beta2-style paging operations. The interface will be available for other API calls to implement as well (i.e. the Audit interface)

5.10.1 Output

Minimal Reports

In the previous interface, reports were templated into schemas for output as specified by the Indivo Reporting Schema. This output method will be preserved for queries that return sets of fact objects, but for queries that return aggregates or groups, we will output data according to the Indivo Aggregate Report Schema.

Generic Reports

Non-Aggregate Generic Reports provide SDMJ or SDMX output based on the requested response format. Please see the documentation for response formats for more information.

Aggregate Also based on the requested response format

- XML
  Formatted according to the Indivo Generic Aggregate Reports Schema

- JSON
  Formatted as SDMJ with AggregateReport as the data model and without a __documentid__. Below is an example of retrieving the max value using the date_group operator

```json
[
  {
    "__modelname__": "AggregateReport",
    "group": "2009-07",
    "value": 1
  }, {
    "__modelname__": "AggregateReport",
    "group": "2009-08",
    "value": 4
  }
]```
5.10.2 Data Fields

As in order_by in the Beta2 interface, each report must expose a set of data fields on which they may be filtered, grouped, or ordered. These fields are data-model dependent, and are explained below.

5.10.3 Query Operators

Filtering Operators

- offset, limit: Syntax is: ?offset={offset}&limit={limit}. These operators will function as previously, taking integer indexes into the result set, and returning a sliced portion of the result set from indices offset to offset + limit.

- Custom Filters: Syntax is: ?{field}={value[|value]...}. Limits result sets to items where the passed field is in the set of pipe delimited values. If no items have such a value in the passed field, the query will return an empty result set. Field names must be data fields exposed by the desired report type.

Ordering Operators

- order_by: Syntax is ?order_by={field}. Functions as previously. Takes a data field exposed by the desired report type, and returns the result set sorted by that field. Fields are sorted in ascending order by default, and prefixing them with a '-' will reverse the order to descending.

  Note: If order_by is used with a grouping, {field} may only refer to the field used with group_by, date_group, or aggregate_by.

Grouping and Aggregating Operators

Note: Calls using grouping and aggregating operators will return data according to the aggregation schema, not the standard query schema

- group_by: Syntax is ?group_by={field}. Groups result sets by the passed field, which must be a data field exposed by the desired report type. Must be used with an aggregation operator, and will throw a 400 Bad Request error otherwise. If used with order_by, the ordering field must be identical to the grouping field or the field passed in aggregate_by.

- aggregate_by: Syntax is ?aggregate_by={operator}*{field}. Combines multiple items in a result set (or a group within a result set) into a single item using the passed operator applied to the passed field (which must be a data field exposed by the desired report type). See below for examples. Available operators are:

  - sum: returns the sum of all values of {field}. Will throw a 400 Bad Request error if the passed field does not contain numerical data.
  - avg: returns the arithmetic mean of all values of {field}. Will throw a 400 Bad Request error if the passed field does not contain numerical data.
  - max: returns the maximum value of all values of {field}. Will throw a 400 Bad Request error if the passed field does not contain numerical data or date/time data.
– min: returns the minimum value of all values of `{field}`. Will throw a 400 Bad Request error if the passed field does not contain numerical data or date/time data.

– count: returns the total number of items passed. If `{field}` is specified, only counts rows where `<tt>{field}</tt>` is not empty.

### Date-based Operators

- **date_range**: Syntax is `?date_range={field}*{start_date}*{end_date}`. A filtering operation that limits result sets to items with values of `{field}` between `{start_date}` and `{end_date}` (inclusive). If either `{start_date}` or `{end_date}` is not specified, the range will be open-ended. If both are unspecified, the filter will do nothing. `{field}` must be a data field exposed by the desired report type. If `{field}` is not a date/time field, a 400 Bad Request error will be raised. `{start_date}` and `{end_date}` must be entered as valid UTC timestamp strings, as described in Basic Data Formats. See below for examples.

- **date_group**: Syntax is: `?date_group={field}*{time_increment}`. A grouping operator that, rather than grouping by a single field value, forms groups based on common increments of time. Has same restraints of use as `group_by` above, with the additional constraint that `{field}` must be a date/time data field.

  **If used with order_by**, the ordering field must be identical to the grouping field or the field passed in `aggregate_by`.

  **Note**: using this operator will result in the return of an aggregation schema.

  Valid increments are:
  - hour: items are placed in the same group if they occurred within the same hour.
  - day: items are placed in the same group if they occurred within the same day.
  - week: items are placed in the same group if they occurred within the same week.
  - month: items are placed in the same group if they occurred within the same month.
  - year: items are placed in the same group if they occurred within the same year.
  - hourofday: items are placed in the same group if they occurred during the same hour of day (even on separate days).
  - dayofweek: items are placed in the same group if they occurred on the same day of the week (even in separate weeks).
  - weekofyear: items are placed in the same group if they occurred during the same week of the year (indexed from 1-52), even in separate years.
  - monthofyear: items are placed in the same group if they occurred during the same month of the year (indexed from 1-12), even in separate years.

### 5.10.4 Query Operator Evaluation

Query operators are evaluated as follows:

1. filter operators, including `date_range` but excluding `limit` and `offset`, are applied first.
2. If `group_by` or `date_group` is passed, it is evaluated next.
3. `aggregate_by` is evaluated next.
4. `order_by`, `limit` and `offset` are applied.
5. The result set is templated into the standard schema or the aggregated schema as appropriate and returned.
5.10.5 Notes on Aggregation

Aggregation over Indivo medical data types could be very useful in certain cases where the data is known (by an app-developer, who generated the data, say) to be highly structured. For example, consider a ‘Pedometer-Visualizer’ app, which reads in data from an electric pedometer worn by a patient, stores that data as Indivo Measurements, and displays to the patient aggregate views of their steps taken (weekly/daily averages, total miles walked, etc.). This app could take full advantage of aggregation functions such as ‘sum’, ‘avg’, etc. However, there are many cases in Indivo where the data, in spite of conforming to Indivo schemas, is not necessarily clean enough to run these aggregations. Consider the case of lab test results: the schema field is by necessity a string, as not all lab results have numerical values. Thus, an incoming query might assume that it could ask for an ‘average lab result value’, when in fact the data wouldn’t support it. We therefore cannot allow numerical aggregations over fields not explicitly labeled as ‘Number’ types. If such a case is necessary for the app, the appropriate design is for the app to make a non-aggregate query, and then process the results itself (i.e., get all lab result values, and then do some data cleaning to insure that only relevant data points are counted in the averaging).

5.10.6 Default Operator Values

If omitted, the following query operators are assigned default values:

- limit: 100
- offset: 0
- order_by: ‘-created_at’ (the date when the fact object was added to indivo). **Only Applied to Non-aggregate Queries: no default ordering for aggregate queries**
- status: active

5.10.7 Valid Query Fields

With the new pluggable data models, valid query fields are defined by the data models themselves. See the Data Models documentation for a more complete explanation.

5.10.8 Example Queries

Below are a number of sample queries that demonstrate the power of the new interface.

**Get all labs of type ‘Hematology’ within a date range**

GET /records/{record_id}/reports/minimal/labs/?lab_type=Hematology&
date_range=date_measured*2009-05-04T00:00:00Z*2011-03-09T00:00:00Z

**Get all labs of type ‘Hematology’ or ‘Chemistry’**

GET /records/{record_id}/reports/minimal/labs/?lab_type=Hematology|Chemistry

**Get the number of lab results per type over the last year**

GET /records/{record_id}/reports/minimal/labs/?group_by=lab_type&
aggregate_by=count*lab_test_name&date_range=date_measured*2010-03-10T00:00:00Z*
Get the number of Hematology labs per month over the last year, ordered by date

GET /records/{record_id}/reports/minimal/labs/?lab_type=Hematology&
date_group=date_measured*month&aggregate_by=count*lab_type&
order_by=-date_measured&date_range=date_measured*2010-03-10T00:00:00Z*

5.11 Connecting Indivo to a Hospital

To connect Indivo to a hospital data feed, you need:

- A hospital that provides an appropriate interface we like to call “Point-of-Care API” (we’ve previously called this Hospital API, but it’s not just hospitals.)
- An Indivo app that reads that data feed and pushes the data into Indivo.

To help this along, we’ve produced:

- the Indivo app in question, which we call Indivo Hospital Connector, and
- a “Mock Hospital”, effectively a fake hospital that implements the Point-of-Care API we think most hospitals should provide.

Once a point of care exposes the Point of Care API, it should then be possible for any PCHR (Indivo, Google Health, Microsoft HealthVault) to connect to the Hospital and lead a user through the data-download process. Of course, this process requires “patient identity proofing”, where the point-of-care will ensure that the person connecting is accessing their data alone.
5.11.1 PoC API

The PoC API is REST API with oAuth used to generate an access token that can then be used for long-term access to the Point-of-care record that needs to be integrated with a PCHR. In every oAuth transaction, the user must authenticate to the data source and authorize access to the connector application (all components are diagrammed above.) Each Point of Care can authenticate users in the way it sees fit: with an account, with one-time credentials, with personal questions only the right patient can answer, etc... this decision is orthogonal to the oAuth process and does not need to be specified for a connector app to function properly. The authentication process, whatever it is, should be implemented as part of the Point of Care’s oAuth log-in-and-authorize process. In addition, each Point of Care can choose which connector apps it wants to enable, by providing only those approved connector apps with oAuth consumer tokens. This means that a Point of Care can implement the PoC API, but only allow specific PCHR’s to connect, once proper privacy/HIPAA arrangements have been made with each. In other words, the protocols are independent of the policy decisions regarding which apps can connect and how users are authenticated: points of care retain full control at all
At the end of the oAuth process, a PoC is expected to return the normal oAuth parameters: oauth_token and oauth_token_secret, and an additional parameter xoauth_hospital_record_id, which identifies the hospital record that has been connected to the token in question. This record ID is needed to make the appropriate REST call to obtain data.

In the PoC API, data is provided in individual documents. The API is then really simple: an Atom feed of documents, and a REST call for each document. Each Point of Care can choose which data format it supports. We encourage Points of Care to use the Indivo Schemas.

GET /data/{hospital_record_id}/documents/
{ atom feed of documents }

GET /data/{hospital_record_id}/documents/{document_id}
{ a single document of data, using identifiers provided in the atom feed }

### 5.11.2 Mock Hospital

The Mock Hospital software we provide implements the PoC REST API. Patient identity proofing is done using the patient’s last name and a confirmation code. In a real-world setting, this confirmation code could be provided in person to the patient, or by mail. It might be preferable to ask the patient to enter more than just their last name, too, and to lock down access if too many incorrect confirmation codes are entered. We do not concern ourselves with this for now: we simply show feasibility of patient-proofing during the oAuth process.

- Download MockHospital v0.1
- Untar in /web/mock_hospital
- Like Indivo, create a PostgreSQL database, preferably called mockhospital, with appropriate PG username access.
- Load the data:
  ../reset.sh
  This will load up data included in the data/ directory, which you can edit to your preference before setting up mockhospital.
- Check out /web/mock_hospital/hospital/dataimport.py to understand the XML and file structure format of the data/ directory
- Run the service:
  python manage.py runserver 7000

### 5.11.3 Indivo Hospital Connector

Indivo Hospital Connector is an Indivo app that also functions as an oAuth client to Mock Hospital (or to any PoC-API-compliant endpoint).

- Download Indivo Hospital Connector
- Set up in /web/indivo_hospital_connector
- Check settings.py for the correct connectivity parameters to Mock Hospital and Indivo.
- Run ./reset.sh to create and initialize the “connector” database and database user
- Run the app:
python manage.py runserver 8003

- Periodically (cron) run the following command to load docs from Mock Hospital to Indivo:
  python manage.py load_documents

5.12 Documenting Indivo

Indivo’s documentation (as of version 1.0) is written using the Sphinx documentation framework, and is hosted by Read the Docs. Additionally, we’ve added some mechanisms to autogenerate documentation for our Indivo API and python client.

5.12.1 Modifying the Docs

If you’ve changed the behavior of the Indivo code base (by adding or modifying API calls) or want to amend some of the language in our documentation, you’ll want to submit changes to the Indivo documentation. Here we describe the mechanisms by which you can alter the documentation.

Autogenerated Docs

After adding, removing, or changing Indivo API calls, you should update our autogenerated documentation to match. You can do that by:

- In indivo_server/, run python manage.py generate_docs parse. This will pull in the latest Indivo code to create an up-to-date skeleton listing of all API calls supported by Indivo. This skeleton is written to indivo_server/doc/sphinx/autogen/api-skeleton.py. If you’ve made changes to the codebase, you should see output from this command listing the API calls that have changed.

- Open up the skeleton file (indivo_server/doc/sphinx/autogen/api-skeleton.py) with the editor of your choice. You’ll see a large python dictionary listing every call that Indivo supports. Simply find the call you’ve modified (by searching for the path of the call, i.e. "/accounts/{ACCOUNT_EMAIL}").

- Update the fields that we couldn’t autogenerate, including:
  - url_params: The parameters passed in via the URL. These will be autopopulated with a default description for common parameters, but you should modify them to be appropriately descriptive, i.e.:
    "url_params": {
      'ACCOUNT_EMAIL': 'The email identifier of the Indivo Account used by the call.'
    },
  - query_opts: Available options to pass in via the querystring of the URL. This should be a dictionary, where keys correspond to valid parameters, and values are descriptions of the parameters, i.e.:
    "query_opts": {
      'order_by': 'The field by which to order results. Available options are "date."'
    },
  - data_fields: Available options to pass in via the POST data of the request. This should be a dictionary, where keys correspond to valid parameters, and values are descriptions of the parameters, i.e.:
    "data_fields": {
      'record_id': 'The Indivo record identifier to associate the data with.'
    },
If there are no parameters in the POST data, but simply raw data, use the key ‘’. For example, the documentation for the call to post a new document looks like:

```json
"data_fields": {
    ‘’: 'The raw content of the document to create.'
},
```

- **return_desc**: A description of the data returned by the call.
- **return_ex**: An example of a return value for the call.
- **deprecated, added, and changed**: Indicators of when the call was added, modified, or deprecated, if applicable. Entries should be the form of python tuples containing the version where the change occurred, and a message describing the change (or the empty string if no message is appropriate). For example, "added": ('2.0.0', '') describes a call added in version 2.0, and "deprecated": ('2.0.0', 'Use call foobar instead') describes a call that has been deprecated in version 2.0, and should be replaced by the foobar call.

**Note**: Some of the fields that are autogenerated (such as “description” and “access_doc”) are pulled directly from the python docstrings of code functions. Make sure that all functions you write in the code have a concise, one-line description as the first line of their docstring.

**Other Changes to the Docs**

If you’d like to make other changes to the docs (say, add a new page describing some functionality or change the language to be more accurate in an existing page), you should simply modify the reStructured Text source, located in `indivo_server/doc/sphinx/source`. For a good primer on writing documentation in reStructured Text, see The Sphinx team’s overview.

**5.12.2 Building the Docs Locally**

Once you’ve made the desired changes to the documentation, you can (and should) test that the docs compile and look appropriate locally before submitting your changes to the project. To do so,

- Run `python manage.py generate_docs build` from the `indivo_server/` directory, which will build the autogenerated docs into an API reference and then build the Sphinx project itself. Currently, you should see a few warning messages when the docs build: that’s fine. Make sure you haven’t introduced any errors.
- Open (in a browser) the compiled files, which are located in `indivo_server/doc/sphinx/build/html`. You’ll see a local copy of the documentation, which you can check to see if your modifications worked as expected. Note that the styles of the local documentation aren’t as pretty as they are on our public readthedocs documentation: we just threw up something basic locally. You didn’t do anything wrong, though—we promise.
- Continue to iterate, modifying the docs, then running `generate_docs build`, until the results are to your satisfaction.

**5.12.3 Getting Your Changes into the Official Docs**

After you’ve made changes to the docs, and tested them locally, you should submit them to the Indivo team so that we can pull them into the official copy. Please submit a Github Pull Request to our Indivo Server repository, and we’ll review your changes and add them to the project documentation.
5.13 Indivo Data Models

5.13.1 Introduction

Data Models in Indivo describe the format in which Indivo represents medical information. They are NOT the same as Schemas, which describe formats that Indivo recognizes as valid input data. Rather, data models describe the final processed state of medical data in Indivo: how data are stored, how they are queryable via the Query API, and how they are returned via the Reporting API.

We also introduce one additional term: Medical Facts. A Fact is one datapoint corresponding to a data model: for example, a latex allergy is a Fact that is an instance of the Allergy data model. Internally, Indivo represents facts as Python objects, so you’ll see us referencing medical facts as fact objects as well.

5.13.2 Defining a Data Model

At its most basic level, a data model definition is just a list of fields and their types. For example, our Problem data model is defined as (some fields omitted):

- date_onset: Date
- date_resolution: Date
- name: String
- comments: String
- diagnosed_by: String

This is pretty simple, and we’d like to enable others add new data models to Indivo just as easily. So we currently allow two formats for defining data models:

Django Model Classes

Since our data models are directly mapped to database tables using Django’s ORM, they are most effectively represented as Django Models. Django has a flexible, powerful method for expressing fields as python class attributes, so data models defined in this way can harness the full capabilities of the Django ORM. Of course, representing data models in this way requires some knowledge of python. For a full reference of Django models, see Django models and Django model fields.

One important Indivo-specific note: when defining Django Model Classes, make sure to subclass indivo.models.Fact, which will ensure that your class can be treated as a data model. For example, your class definition might look like:

```python
from indivo.models import Fact
from django.db import models

class YourModel(Fact):
    your_field1 = models.CharField(max_length=200, null=True)

    ... # Additional fields here
```
Custom Django Model Fields

For modeling medical data, Indivo provides some custom Field Subclasses. These fields represent their data as multiple separate database fields, with names formed from the original field’s name and some appended suffixes (see the classes below for some examples). You should use these fields as if they were any other Django Model Field:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import YourFavoriteFieldSubclass

class YourModel(Fact):
    normal_field = models.CharField(max_length=200, null=True)
    special_field = YourFavoriteFieldSubclass()
```

Now YourModel has both a standard CharField, and also other fields defined by the Field Subclass. We define the following Field Subclasses:

```python
class indivo.fields.CodedValueField(Type)
    A field for representing coded data elements.
    Creating a CodedValueField named ‘value’, for example, will (under the hood) create thee fields:
    • value_identifier, the system-specific identifier that represents the element (i.e. an RXNorm CUI)
    • value_title, the human-readable title of the element
    • value_system, the coding system used to represent the element
    When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original value field name.

class indivo.fields.ValueAndUnitField(Type)
    A field for representing data elements with both a value and a unit.
    Creating a ValueAndUnitField named ‘frequency’, for example, will (under the hood) create the fields:
    • frequency_value, the value of the element
    • frequency_unit, the units in which the value is measured
    When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original frequency field name.

class indivo.fields.AddressField(Type)
    A field for representing a physical address.
    Creating an AddressField named ‘address’, for example, will (under the hood) create the fields:
    • address_country, the country in which the address is located
    • address_city, the city in which the address is located
    • address_postalcode, the postalcode of the address
    • address_region, the region (state, in the US) in which the address is located
    • address_street, the street address (including street number, apartment number, etc.) at which the address is located
    When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original address field name.

class indivo.fields.NameField(Type)
    A field for representing a person’s name.
Creating a NameField named ‘name’, for example, will (under the hood) create the fields:

- `name_family`, the family (last) name of the person
- `name_given`, the given (first) name of the person
- `name_middle`, the middle name of the person
- `name_prefix`, the prefix (i.e. ‘Mr.’, ‘Sir’, etc.) for the person’s name
- `name_suffix`, the suffix (i.e. ‘Jr.’, ‘Ph.D.’, etc.) for the person’s name

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `name` field name.

```python
class indivo.fields.TelephoneField(Type)
A field for representing a telephone number.

Creating a TelephoneField named ‘phone’, for example, will (under the hood) create the fields:

- `phone_type`, The type of the phone number, limited to `h` (home), `w` (work), or `c` (cell)
- `phone_number`, The actual phone number
- `phone_preferred_p`, Whether or not this number is a preferred method of contact (True or False)

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `phone` field name.

```python
class indivo.fields.PharmacyField(Type)
A field for representing a pharmacy.

Creating a PharmacyField named ‘pharmacy’, for example, will (under the hood) create three fields:

- `pharmacy_ncpdpid`, the pharmacy’s National Council for Prescription Drug Programs (NCPDP) ID number
- `pharmacy_adr`, the address at which the pharmacy is located (an AddressField)
- `pharmacy_org`, the name of the organization that owns the pharmacy

When describing instances of your model (either when defining a transform output or when referencing fields using the Indivo Query API), you must refer to these field names, not the original `pharmacy` field name.

```python
class indivo.fields.ProviderField(Type)
A field for representing a medical provider.

Creating a ProviderField named ‘doc’, for example, will (under the hood) create the fields:

- `doc_dea_number`, the provider’s Drug Enforcement Agency (DEA) number
- `doc_ethnicity`, the provider’s ethnicity
- `doc_npi_number`, the provider’s National Provider Identification (NPI) number
- `doc_preferred_language`, the provider’s preferred language
- `doc_race`, the provider’s race
- `doc_adr`, the provider’s address (an AddressField)
- `doc_bday`, the provider’s birth date
- `doc_email`, the provider’s email address
- `doc_name`, the provider’s name (a NameField)
- `doc_tel_1`, the provider’s primary phone number (a TelephoneField)`
• `doc_tel_2`, the provider’s secondary phone number (a `TelephoneField`)

• `doc_gender`, the provider’s gender, limited to m (male) or f (female)

When describing instances of your model (either when defining a `transform output` or when referencing fields using the `Indivo Query API`), you must refer to these field names, not the original `doc` field name.

class `indivo.fields.VitalSignField` *(Type)*

A field for representing a single measurement of a vital sign.

Creating a `VitalSignField` named ‘bp’, for example, will (under the hood) create the fields:

• `bp_unit`, the unit of the measurement

• `bp_value`, the value of the measurement

• `bp_name`, the name of the measurement (a `CodedValueField`)

When describing instances of your model (either when defining a `transform output` or when referencing fields using the `Indivo Query API`), you must refer to these field names, not the original `bp` field name.

class `indivo.fields.BloodPressureField` *(Type)*

A field for representing a blood pressure measurement.

Creating a `BloodPressureField` named ‘bp’, for example, will (under the hood) create the fields:

• `bp_position`, the position in which the measurement was taken (a `CodedValueField`)

• `bp_site`, the site on the body at which the measurement was taken (a `CodedValueField`)

• `bp_method`, the method of the measurement (a `CodedValueField`)

• `bp_diastolic`, the diastolic blood pressure (a `VitalSignField`)

• `bp_systolic`, the systolic blood pressure (a `VitalSignField`)

When describing instances of your model (either when defining a `transform output` or when referencing fields using the `Indivo Query API`), you must refer to these field names, not the original `bp` field name.

class `indivo.fields.ValueRangeField` *(Type)*

A field for representing a range of values.

Creating a `ValueRangeField` named ‘normal_range’, for example, will (under the hood) create the fields:

• `normal_range_max`, the maximum value of the range (a `ValueAndUnitField`)

• `normal_range_min`, the minimum value of the range (a `ValueAndUnitField`)

When describing instances of your model (either when defining a `transform output` or when referencing fields using the `Indivo Query API`), you must refer to these field names, not the original `normal_range` field name.

class `indivo.fields.QuantitativeResultField` *(Type)*

A field for representing a quantitative result, and expected ranges for that result.

Creating a `QuantitativeResultField` named ‘lab_result’, for example, will (under the hood) create the fields:

• `lab_result_non_critical_range`, the range outside of which results are ‘critical’ (a `ValueRangeField`)

• `lab_result_normal_range`, the range outside of which results are ‘abnormal’ (a `ValueRangeField`)

• `lab_result_value`, the actual result (a `ValueAndUnitField`)

When describing instances of your model (either when defining a `transform output` or when referencing fields using the `Indivo Query API`), you must refer to these field names, not the original `lab_result` field name.
**Simple Data Modeling Language (SDML)**

For those less python-savvy who are still capable of thinking in terms of ‘fields’ and ‘types’ (which should be most people), we’ve defined a JSON-based modeling language for defining the very simple data models easily. SDML is less flexible than Django’s modeling language, but is much quicker to get started with and is less verbose for describing simple models. See our documentation of the language [here](#).

**Feeling Lost?**

For help getting started, see our [core data models](#), below, each of which provide definitions both in SDML and Django Model classes.

### 5.13.3 Data Models and the Query API

Since the *Query API* allows app developers to directly apply filters and ranges to the datamodels they are selecting, they need to know what fields they are allowed to query against. The answer is simple:

**ANY FIELD ON A DATA MODEL THAT IS NOT A RELATION TO ANOTHER MODEL MAY BE USED IN THE QUERY API!**

For example, we introduced the ‘Problem’ model above, which has the fields:

- *date_onset*: Date
- *date_resolution*: Date
- *name*: String
- *comments*: String
- *diagnosed_by*: String

If you were making an API call such as `GET /records/(RECORD_ID)/reports/minimal/problems/`, you could filter by any of:

- *date_onset*
- *date_resolution*
- *name*
- *comments*
- *diagnosed_by*

If the problems model were a bit more complicated, and had another field:

- *prescribed_med*: Medication

You wouldn’t be able to filter by *prescribed_med*, since that field is a relation to another model.

The only exceptions to this rule are *custom Django Model Fields*. Such fields are translated into fields with other names, as described above. Any of these fields may be used in the query API, but (for example), when looking at a model with a CodedValue element such as:

- *problem_type*: CodedValue

You would be able to filter by *problem_type_identifier, problem_type_title, or problem_type_system*, but not by *problem_type* itself.
5.13.4 Core Data Models

Here is a listing of the data models currently supported by Indivo. Each instance might define other, contributed models: see below for information on how to add data models to Indivo.

Indivo Data Model: Allergy

Model Definition

As SDML:

```json
[{
    "__modelname__": "Allergy",
    "allergic_reaction": "CodedValue",
    "category": "CodedValue",
    "drug_allergen": "CodedValue",
    "drug_class_allergen": "CodedValue",
    "food_allergen": "CodedValue",
    "severity": "CodedValue"
},
{
    "__modelname__": "AllergyExclusion",
    "name": "CodedValue"
}]
```

As a Django Model Class:

```python
from indivo.models import Fact
from indivo.fields import CodedValueField

class Allergy(Fact):
    allergic_reaction = CodedValueField()
    category = CodedValueField()
    drug_allergen = CodedValueField()
    drug_class_allergen = CodedValueField()
    food_allergen = CodedValueField()
    severity = CodedValueField()

class AllergyExclusion(Fact):
    name = CodedValueField()
```

Examples

As SDMJ:

```json
[{
    "__modelname__": "Allergy",
    "allergic_reaction_title": "Anaphylaxis",
    "allergic_reaction_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
    "allergic_reaction_identifier": "39579001",
    "category_title": "Drug allergy",
    "category_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
    "category_identifier": "416098002",
    "drug_class_allergen_title": "Sulfonamide Antibacterial",
    "drug_class_allergen_system": "http://purl.bioontology.org/ontology/NDFRT/",
    "drug_class_allergen_identifier": "N0000175503",
```
As SDMX:

```xml
<Models>
  <Model name="Allergy">
    <Field name="allergic_reaction_title">Anaphylaxis</Field>
    <Field name="allergic_reaction_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="allergic_reaction_identifier">39579001</Field>
    <Field name="category_title">Drug allergy</Field>
    <Field name="category_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="category_identifier">416098002</Field>
    <Field name="drug_class_allergen_title">Sulfonamide Antibacterial</Field>
    <Field name="drug_class_allergen_system">http://purl.bioontology.org/ontology/NDFRT/</Field>
    <Field name="drug_class_allergen_identifier">N0000175503</Field>
    <Field name="severity_title">Severe</Field>
    <Field name="severity_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
    <Field name="severity_identifier">24484000</Field>
  </Model>
  <Model name="AllergyExclusion">
    <Field name="name_title">No known allergies</Field>
    <Field name="name_identifier">160244002</Field>
    <Field name="name_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
  </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import Allergy

allergy_fact = Allergy(
    allergic_reaction_title="Anaphylaxis",
    allergic_reaction_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    allergic_reaction_identifier="39579001",
    category_title="Drug allergy",
    category_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    category_identifier="416098002",
    drug_class_allergen_title="Sulfonamide Antibacterial",
    drug_class_allergen_system="http://purl.bioontology.org/ontology/NDFRT/",
    drug_class_allergen_identifier="N0000175503",
    severity_title="Severe",
    severity_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    severity_identifier="24484000",
)

allergy_exclusion = AllergyExclusion(
    name_title="No known allergies",
    name_identifier="160244002",
    name_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
)```
Indivo Data Model: Equipment

Model Definition

As SDML:

```
{  
    "__modelname__": "Equipment",
    "date_started": "Date",
    "date_stopped": "Date",
    "name": "String",
    "vendor": "String",
    "description": "String"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class Equipment(Fact):
    date_started = models.DateField(null=True)
    date_stopped = models.DateField(null=True)
    name = models.CharField(max_length=40)
    vendor = models.CharField(max_length=40, null=True)
    description = models.TextField(null=True)
```

Examples

As SDMJ:

```
{  
    "__modelname__": "Equipment",
    "date_started": "2009-02-05",
    "date_stopped": "2009-06-12",
    "name": "Pacemaker",
    "vendor": "Acme Medical Devices",
    "description": "it works!"
}
```

As SDMX:

```xml
<Models>
    <Model name="Equipment">
        <Field name="date_started">2009-02-05</Field>
        <Field name="date_stopped">2009-06-12</Field>
        <Field name="name">Pacemaker</Field>
        <Field name="vendor">Acme Medical Devices</Field>
        <Field name="description">it works!</Field>
    </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import Equipment
from indivo.lib.iso8601 import parse_utc_date as date

equipment_fact = Equipment(
```
Indivo Data Model: Immunization

Model Definition

As SDML:

```json
{
  "__modelname__": "Immunization",
  "date": "Date",
  "administration_status": "CodedValue",
  "product_class": "CodedValue",
  "product_class_2": "CodedValue",
  "product_name": "CodedValue",
  "refusal_reason": "CodedValue"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField

class Immunization(Fact):
    date = models.DateTimeField(null=True)
    administration_status = CodedValueField()
    product_class = CodedValueField()
    product_class_2 = CodedValueField()
    product_name = CodedValueField()
    refusal_reason = CodedValueField()
```

Examples

As SDMJ:

```json
{
  "__modelname__": "Immunization",
  "date": "2009-05-16T12:00:00Z",
  "administration_status_title": "Not Administered",
  "administration_status_system": "http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#",
  "administration_status_identifier": "notAdministered",
  "product_class_title": "TYPHOID",
  "product_class_system": "http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#",
  "product_class_identifier": "TYPHOID",
  "product_name_title": "typhoid, oral",
  "product_name_system": "http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx#",
  "product_name_identifier": "25",
  "refusal_reason_title": "Allergy to vaccine/vaccine components, or allergy to eggs",
  "refusal_reason_system": "http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#",
}```
"refusal_reason_identifier": "allergy"
}

As SDMX:

<Models>
  <Model name="Immunization">
    <Field name="date">2009-05-16T12:00:00Z</Field>
    <Field name="administration_status_title">Not Administered</Field>
    <Field name="administration_status_system">http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#</Field>
    <Field name="administration_status_identifier">notAdministered</Field>
    <Field name="product_class_title">TYPHOID</Field>
    <Field name="product_class_system">http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#</Field>
    <Field name="product_class_identifier">TYPHOID</Field>
    <Field name="product_name_title">typhoid, oral</Field>
    <Field name="product_name_system">http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx#</Field>
    <Field name="product_name_identifier">25</Field>
    <Field name="refusal_reason_title">Allergy to vaccine/vaccine components, or allergy to eggs</Field>
    <Field name="refusal_reason_system">http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#</Field>
    <Field name="refusal_reason_identifier">allergy</Field>
  </Model>
</Models>

As a Fact object:

from indivo.models import Immunization
from indivo.lib.iso8601 import parse_utc_date as date

immunization_fact = Immunization(
    date=date("2009-05-16T12:00:00Z"),
    administration_status_title="Not Administered",
    administration_status_system="http://smartplatforms.org/terms/codes/ImmunizationAdministrationStatus#",
    administration_status_identifier="notAdministered",
    product_class_title="TYPHOID",
    product_class_system="http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=vg#",
    product_class_identifier="TYPHOID",
    product_name_title="typhoid, oral",
    product_name_system="http://www2a.cdc.gov/nip/IIS/IISStandards/vaccines.asp?rpt=cvx#",
    product_name_identifier="25",
    refusal_reason_title="Allergy to vaccine/vaccine components, or allergy to eggs",
    refusal_reason_system="http://smartplatforms.org/terms/codes/ImmunizationRefusalReason#",
    refusal_reason_identifier="allergy",
)

Indivo Data Model: LabResult

Model Definition

As SDML:

{  
  "__modelname__": "LabResult",
  "abnormal_interpretation": "CodedValue",
  "accession_number": "String",
  "test_name": "CodedValue",
  "status": "CodedValue",
  "narrative_result": "String",
}
"notes": "String",
"quantitative_result": "QuantitativeResult",
"collected_at": "Date",
"collected_by_org": "Organization",
"collected_by_name": "Name",
"collected_by_role": "String"
}

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField, QuantitativeResultField, OrganizationField, NameField

class LabResult(Fact):
    abnormal_interpretation = CodedValueField()
    accession_number = models.CharField(max_length=255, null=True)
    test_name = CodedValueField()
    status = CodedValueField()
    narrative_result = models.CharField(max_length=255, null=True)
    notes = models.CharField(max_length=600, null=True)
    quantitative_result = QuantitativeResultField()
    collected_at = models.DateTimeField(null=True)
    collected_by_org = OrganizationField()
    collected_by_name = NameField()
    collected_by_role = models.CharField(max_length=255, null=True)
```

Examples

As SDMJ:

```json
{
    "__modelname__": "LabResult",
    "abnormal_interpretation_title": "Normal",
    "abnormal_interpretation_system": "http://smartplatforms.org/terms/codes/LabResultInterpretation#",
    "abnormal_interpretation_identifier": "normal",
    "accession_number": "AC09205823577",
    "test_name_title": "Serum Sodium",
    "test_name_system": "http://purl.bioontology.org/ontology/LNC/",
    "test_name_identifier": "2951-2",
    "status_title": "Final results": complete and verified",
    "status_system": "http://smartplatforms.org/terms/codes/LabStatus#",
    "status_identifier": "final",
    "notes": "Blood sample appears to have hemolyzed",
    "quantitative_result_non_critical_range_max_value": "155",
    "quantitative_result_non_critical_range_max_unit": "mEq/L",
    "quantitative_result_non_critical_range_min_value": "120",
    "quantitative_result_non_critical_range_min_unit": "mEq/L",
    "quantitative_result_normal_range_max_value": "145",
    "quantitative_result_normal_range_max_unit": "mEq/L",
    "quantitative_result_normal_range_min_value": "135",
} 
```
As SDMX:

```xml
<Models>
  <Model name="LabResult">
    <Field name="abnormal_interpretation_title">Normal</Field>
    <Field name="abnormal_interpretation_system">http://smartplatforms.org/terms/codes/LabResultInterpretation#</Field>
    <Field name="abnormal_interpretation_identifier">normal</Field>
    <Field name="accession_number">AC09205823577</Field>
    <Field name="test_name_title">Serum Sodium</Field>
    <Field name="test_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
    <Field name="test_name_identifier">2951-2</Field>
    <Field name="status_title">Final results: complete and verified</Field>
    <Field name="status_system">http://smartplatforms.org/terms/codes/LabStatus#</Field>
    <Field name="status_identifier">final</Field>
    <Field name="notes">Blood sample appears to have hemolyzed</Field>
    <Field name="quantitative_result_non_critical_range_max_value">155</Field>
    <Field name="quantitative_result_non_critical_range_max_unit">mEq/L</Field>
    <Field name="quantitative_result_non_critical_range_min_value">120</Field>
    <Field name="quantitative_result_non_critical_range_min_unit">mEq/L</Field>
    <Field name="quantitative_result_normal_range_max_value">145</Field>
    <Field name="quantitative_result_normal_range_max_unit">mEq/L</Field>
    <Field name="quantitative_result_normal_range_min_value">135</Field>
    <Field name="quantitative_result_normal_range_min_unit">mEq/L</Field>
    <Field name="quantitative_result_value_value">140</Field>
    <Field name="quantitative_result_value_unit">mEq/L</Field>
    <Field name="collected_at">2010-12-27T17:00:00Z</Field>
    <Field name="collected_by_org_name">City Lab</Field>
    <Field name="collected_by_org_adr_country">USA</Field>
    <Field name="collected_by_org_adr_city">Springfield</Field>
    <Field name="collected_by_org_adr_postalcode">11111</Field>
    <Field name="collected_by_org_adr_region">MA</Field>
    <Field name="collected_by_org_adr_street">20 Elm St</Field>
    <Field name="collected_by_name_family">Finnialispi</Field>
    <Field name="collected_by_name_given">Tad</Field>
    <Field name="collected_by_role">"Lab Specialist"</Field>
  </Model>
</Models>
```
As a Fact object:

```python
from indivo.models import LabResult
from indivo.lib.iso8601 import parse_utc_date as date

lab_fact = LabResult(
    abnormal_interpretation_title="Normal",
    abnormal_interpretation_system="http://smartplatforms.org/terms/codes/LabResultInterpretation#",
    abnormal_interpretation_identifier="normal",
    accession_number="AC09205823577",
    test_name_title="Serum Sodium",
    test_name_system="http://purl.bioontology.org/ontology/LNC/",
    test_name_identifier="2951-2",
    status_title="Final results: complete and verified",
    status_system="http://smartplatforms.org/terms/codes/LabStatus#",
    status_identifier="final",
    notes="Blood sample appears to have hemolyzed",
    quantitative_result_non_critical_range_max_value="155",
    quantitative_result_non_critical_range_max_unit="mEq/L",
    quantitative_result_non_critical_range_min_value="120",
    quantitative_result_non_critical_range_min_unit="mEq/L",
    quantitative_result_normal_range_max_value="145",
    quantitative_result_normal_range_max_unit="mEq/L",
    quantitative_result_normal_range_min_value="135",
    quantitative_result_normal_range_min_unit="mEq/L",
    quantitative_result_value_value="140",
    quantitative_result_value_unit="mEq/L",
    collected_at=date("2010-12-27T17:00:00Z"),
    collected_by_org_name="City Lab",
    collected_by_org_adr_country="USA",
    collected_by_org_adr_city="Springfield",
    collected_by_org_adr_postalcode="11111",
    collected_by_org_adr_region="MA",
    collected_by_org_adr_street="20 Elm St",
    collected_by_name_family="Finnialispi",
    collected_by_name_given="Tad",
    collected_by_role="Lab Specialist",
)
```

5.13. Indivo Data Models
Indivo Data Model: Medication

Model Definition

As SDML:

```
{   "__modelname__": "Medication",
   "drugName": "CodedValue",
   "endDate": "Date",
   "frequency": "ValueAndUnit",
   "instructions": "String",
   "provenance": "CodedValue",
   "quantity": "ValueAndUnit",
   "startDate": "Date",
   "fulfillments": [{
      "__modelname__": "Fill",
      "date": "Date",
      "dispenseDaysSupply": "Number",
      "pbm": "String",
      "pharmacy": "Pharmacy",
      "provider": "Provider",
      "quantityDispensed": "ValueAndUnit"
   }]
}
```

Note: Since SDML doesn’t provide for Boolean Fields, we are unable to define the `dispense_as_written` field properly in SDML. Our actual implementation of the Medication data model uses a Django Model Class for this reason.

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import CodedValueField, ValueAndUnitField, PharmacyField, ProviderField

class Medication(Fact):
    drugName = CodedValueField()
    endDate = models.DateField(null=True)
    frequency = ValueAndUnitField()
    instructions = models.CharField(max_length=255, null=True)
    provenance = CodedValueField()
    quantity = ValueAndUnitField()
    startDate = models.DateField(null=True)

class Fill(Fact):
    date = models.DateTimeField(null=True)
    dispenseDaysSupply = models.FloatField(null=True)
    pbm = models.CharField(max_length=255, null=True)
    pharmacy = PharmacyField()
    provider = ProviderField()
    quantityDispensed = ValueAndUnitField()
    medication = models.ForeignKey(Medication, null=True, related_name='fulfillments')
```
Examples

As SDMJ:

```
{
  "__modelname__": "Medication",
  "drugName_title": "AMITRIPTYLINE HCL 50 MG TAB",
  "drugName_system": "http://purl.bioontology.org/ontology/RXNORM/",
  "drugName_identifier": "856845",
  "endDate": "2007-08-14",
  "frequency_value": "2",
  "frequency_unit": "/d",
  "instructions": "Take two tablets twice daily as needed for pain",
  "provenance_title": "Derived by prescription",
  "provenance_system": "http://smartplatforms.org/terms/codes/MedicationProvenance#",
  "provenance_identifier": "prescription",
  "quantity_value": "2",
  "quantity_unit": "{tablet}",
  "startDate": "2007-03-14",
  "fulfillments": [
    {
      "__modelname__": "Fill",
      "date": "2007-03-14T04:00:00Z",
      "dispenseDaysSupply": "30",
      "pbm": "T00000000001011",
      "pharmacy_ncpdpid": "5235235",
      "pharmacy_org": "CVS #588",
      "pharmacy_adr_country": "Australia",
      "pharmacy_adr_city": "WonderCity",
      "pharmacy_adr_postalcode": "5555",
      "pharmacy_adr_street": "111 Lake Drive",
      "provider_dea_number": "3255555555",
      "provider_npi_number": "5235235",
      "provider_email": "joshua.mandel@fake.emailserver.com",
      "provider_name_given": "Josuha",
      "provider_name_family": "Mandel",
      "provider_tel_1_type": "w",
      "provider_tel_1_number": "1-235-947-3452",
      "provider_tel_1_preferred_p": true,
      "quantityDispensed_value": "60",
      "quantityDispensed_unit": "{tablet}",
    },
    {
      "__modelname__": "Fill",
      "date": "2007-04-14T04:00:00Z",
      "dispenseDaysSupply": "30",
      "pbm": "T00000000001011",
      "pharmacy_ncpdpid": "5235235",
      "pharmacy_org": "CVS #588",
      "pharmacy_adr_country": "Australia",
      "pharmacy_adr_city": "WonderCity",
      "pharmacy_adr_postalcode": "5555",
      "pharmacy_adr_street": "111 Lake Drive",
      "provider_dea_number": "3255555555",
      "provider_npi_number": "5235235",
      "provider_email": "joshua.mandel@fake.emailserver.com",
      "provider_name_given": "Josuha",
      "provider_name_family": "Mandel",
    }
  ],
}
```
As SDMX:

<Models>
    <Model name="Medication">
        <Field name="drugName_title">AMITRIPTYLINE HCL 50 MG TAB</Field>
        <Field name="drugName_system">http://purl.bioontology.org/ontology/RXNORM/</Field>
        <Field name="drugName_identifier">856845</Field>
        <Field name="endDate">2007-08-14</Field>
        <Field name="frequency_value">2</Field>
        <Field name="frequency_unit">/d</Field>
        <Field name="instructions">Take two tablets twice daily as needed for pain</Field>
        <Field name="provenance_title">Derived by prescription</Field>
        <Field name="provenance_system">http://smartplatforms.org/terms/codes/MedicationProvenance#</Field>
        <Field name="provenance_identifier">prescription</Field>
        <Field name="quantity_value">2</Field>
        <Field name="quantity_unit">{tablet}</Field>
        <Field name="startDate">2007-03-14</Field>
        <Field name="fulfillments">
            <Models>
                <Model name="Fill">
                    <Field name="date">2007-03-14T04:00:00Z</Field>
                    <Field name="dispenseDaysSupply">30</Field>
                    <Field name="pbm">T00000000001011</Field>
                    <Field name="pharmacy_ncpdpid">5235235</Field>
                    <Field name="pharmacy_org">CVS #588</Field>
                    <Field name="pharmacy_adr_country">Australia</Field>
                    <Field name="pharmacy_adr_city">WonderCity</Field>
                    <Field name="pharmacy_adr_postalcode">5555</Field>
                    <Field name="pharmacy_adr_street">111 Lake Drive</Field>
                    <Field name="provider_dea_number">325555555</Field>
                    <Field name="provider_npi_number">5235235</Field>
                    <Field name="provider_email">joshua.mandel@fake.emailserver.com</Field>
                    <Field name="provider_name_given">Josuha</Field>
                    <Field name="provider_name_family">Mandel</Field>
                    <Field name="provider_tel_1_type">w</Field>
                    <Field name="provider_tel_1_number">1-235-947-3452</Field>
                    <Field name="provider_tel_1_preferred_p">true</Field>
                    <Field name="quantityDispensed_value">60</Field>
                    <Field name="quantityDispensed_unit">{tablet}</Field>
                </Model>
                <Model name="Fill">
                    <Field name="date">2007-04-14T04:00:00Z</Field>
                    <Field name="dispenseDaysSupply">30</Field>
                    <Field name="pbm">T00000000001011</Field>
                    <Field name="pharmacy_ncpdpid">5235235</Field>
                    <Field name="pharmacy_org">CVS #588</Field>
                    <Field name="pharmacy_adr_country">Australia</Field>
                    <Field name="pharmacy_adr_city">WonderCity</Field>
                    <Field name="pharmacy_adr_postalcode">5555</Field>
                </Model>
            </Models>
        </Field>
    </Model>
</Models>
As a Fact object:

```python
from indivo.models import Medication, Fill
from indivo.lib.iso8601 import parse_utc_date as date

med = Medication(
    drugName_title="AMITRIPTYLINE HCL 50 MG TAB",
    drugName_system="http://purl.bioontology.org/ontology/RXNORM/",
    drugName_identifier="856845",
    endDate=date("2007-08-14"),
    frequency_value="2",
    frequency_unit="/d",
    instructions="Take two tablets twice daily as needed for pain",
    provenance_title="Derived by prescription",
    provenance_system="http://smartplatforms.org/terms/codes/MedicationProvenance#",
    provenance_identifier="prescription",
    quantity_value="2",
    quantity_unit="{tablet}",
    startDate=date("2007-03-14"),
)

fill1 = Fill(
    date=date("2007-03-14T04:00:00Z"),
    dispenseDaysSupply=30,
    pbm="T00000000001011",
    pharmacy_ncpdpid="5235235",
    pharmacy_org="CVS #588",
    pharmacy_adr_country="Australia",
    pharmacy_adr_city="WonderCity",
    pharmacy_adr_postalcode="5555",
    pharmacy_adr_street="111 Lake Drive",
    provider_dea_number="3255555555",
    provider_npi_number="5235235",
    provider_email="joshua.mandel@fake.emailserver.com",
    provider_name_given="Josuha",
    provider_name_family="Mandel",
    provider_tel_1_type="w",
    provider_tel_1_number="1-235-947-3452",
    provider_tel_1_preferred_p=True,
    quantityDispensed_value="60",
    quantityDispensed_unit="{tablet}" )
```
fill2 = Fill(
    date=date("2007-04-14T04:00:00Z"),
    dispenseDaysSupply=30,
    pbm="T00000000001011",
    pharmacy_ncpdpid="5235235",
    pharmacy_org="CVS #588",
    pharmacy_adr_country="Australia",
    pharmacy_adr_city="WonderCity",
    pharmacy_adr_postalcode="5555",
    pharmacy_adr_street="111 Lake Drive",
    provider_dea_number="3255555555",
    provider_npi_number="5235235",
    provider_email="joshua.mandel@fake.emailserver.com",
    provider_name_given="Josuha",
    provider_name_family="Mandel",
    provider_tel_1_type="w",
    provider_tel_1_number="1-235-947-3452",
    provider_tel_1_preferred_p=True,
    quantityDispensed_value="60",
    quantityDispensed_unit="{tablet}";
)
# save the medication so we can relate other objects to it
med.save()
med.fulfillments = [fill1, fill2]
med.save()

Indivo Data Model: Problem

Model Definition

As SDML:

{  
    "__modelname__": "Problem",
    "startDate": "Date",
    "endDate": "Date",
    "name": "CodedValue",
    "notes": "String"
}

As a Django Model Class:

```python
from indivo.models import Fact
from indivo.fields import CodedValueField
from django.db import models

class Problem(Fact):
    startDate = models.DateTimeField(null=True)
    endDate = models.DateTimeField(null=True)
    name = CodedValueField()
    notes = models.TextField(null=True)
```
Examples

As SDMJ:

```json
{
    "__modelname__": "Problem",
    "startDate": "2009-05-16T12:00:00Z",
    "endDate": "2009-05-16T16:00:00Z",
    "name_title": "Backache (finding)",
    "name_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
    "name_identifier": "161891005"
}
```

As SDMX:

```xml
<Models>
    <Model name="Problem">
        <Field name="startDate">2009-05-16T12:00:00Z</Field>
        <Field name="endDate">2009-05-16T16:00:00Z</Field>
        <Field name="name_title">Backache (Finding)</Field>
        <Field name="name_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
        <Field name="name_identifier">161891005</Field>
    </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import Problem
from indivo.lib.iso8601 import parse_utc_date as date

problem_fact = Problem(
    startDate=date("2009-05-16T12:00:00Z"),
    endDate=date("2009-05-16T16:00:00Z"),
    name_title="Backache (finding)",
    name_system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    name_identifier="161891005",
)
```

Indivo Data Model: Procedure

Model Definition

As SDML:

```json
{
    "__modelname__": "Procedure",
    "date_performed": "Date",
    "name": "String",
    "name_type": "String",
    "name_value": "String",
    "name_abbrev": "String",
    "provider_name": "String",
    "provider_institution": "String",
    "location": "String",
    "comments": "String"
}
```
As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class Procedure(Fact):
    date_performed = models.DateTimeField(null=True)
    name = models.CharField(max_length=100)
    name_type = models.CharField(max_length=80, null=True)
    name_value = models.CharField(max_length=40, null=True)
    name_abbrev = models.CharField(max_length=20, null=True)
    provider_name = models.CharField(max_length=200, null=True)
    provider_institution = models.CharField(max_length=200, null=True)
    location = models.CharField(max_length=100, null=True)
    comments = models.TextField(null=True)
```

Examples

As **SDMJ**:

```json
{
    "__modelname__": "Procedure",
    "date_performed": "2009-05-16T12:00:00",
    "name": "Appendectomy",
    "name_type": "http://codes.indivo.org/procedures#",
    "name_value": "123",
    "name_abbrev": "append",
    "provider_name": "Kenneth Mandl",
    "provider_institution": "Children’s Hospital Boston",
    "location": "300 Longwood Ave, Boston MA 02115",
    "comments": "Went great!"
}
```

As **SDMX**:

```xml
<Models>
  <Model name="Procedure">
    <Field name="date_performed">2009-05-16T12:00:00</Field>
    <Field name="name">Appendectomy</Field>
    <Field name="name_type">http://codes.indivo.org/procedures#</Field>
    <Field name="name_value">123</Field>
    <Field name="name_abbrev">append</Field>
    <Field name="provider_name">Kenneth Mandl</Field>
    <Field name="provider_institution">Children’s Hospital Boston</Field>
    <Field name="location">300 Longwood Ave, Boston MA 02115</Field>
    <Field name="comments">Went great!</Field>
  </Model>
</Models>
```

As a **Fact object**:

```python
from indivo.models import Procedure
from indivo.lib.iso8601 import parse_utc_date as date

procedure_fact = Procedure(
    date_performed=date("2009-05-16T12:00:00"),
    name="Appendectomy",
    name_type="http://codes.indivo.org/procedures#",
```
Indivo Data Model: VitalSigns

Model Definition

As SDML:

```json
{
    "__modelname__": "VitalSigns",
    "date": "Date",
    "encounter": {
        "__modelname__": "Encounter",
        "startDate": "Date",
        "endDate": "Date",
        "facility": "Organization",
        "provider": "Provider",
        "encounterType": "CodedValue"
    },
    "bp": "BloodPressure",
    "bmi": "VitalSign",
    "heart_rate": "VitalSign",
    "height": "VitalSign",
    "oxygen_saturation": "VitalSign",
    "respiratory_rate": "VitalSign",
    "temperature": "VitalSign",
    "weight": "VitalSign"
}
```

As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models
from indivo.fields import BloodPressureField, VitalSignField, CodedValueField, OrganizationField, ProviderField

class VitalSigns(Fact):
    date = models.DateTimeField(null=True)
    encounter = models.ForeignKey('Encounter', null=True)
    bp = BloodPressureField()
    bmi = VitalSignField()
    heart_rate = VitalSignField()
    height = VitalSignField()
    oxygen_saturation = VitalSignField()
    respiratory_rate = VitalSignField()
    temperature = VitalSignField()
    weight = VitalSignField()

class Encounter(Fact):
    startDate = models.DateTimeField(null=True)
    endDate = models.DateTimeField(null=True)
    facility = OrganizationField()
```

5.13. Indivo Data Models
provider = ProviderField()
encounterType = CodedValueField()

Examples

As SDMJ:

```
{
  "__modelname__": "VitalSigns",
  "date": "2009-05-16T12:00:00Z",
  "encounter": {
    "__modelname__": "Encounter",
    "startDate": "2009-05-16T12:00:00Z",
    "endDate": "2009-05-16T16:00:00Z",
    "facility_name": "Wonder Hospital",
    "facility adr country": "Australia",
    "facility adr city": "WonderCity",
    "facility adr postalcode": "5555",
    "facility adr street": "111 Lake Drive",
    "provider_dea_number": "3255555555",
    "provider_npi_number": "5235235",
    "provider_email": "joshua.mandel@fake.emailserver.com",
    "provider name given": "Josuha",
    "provider name family": "Mandel",
    "provider tel_1 type": "w",
    "provider tel_1 number": "1-235-947-3452",
    "provider tel_1 preferred_p": true,
    "encounterType title": "Ambulatory encounter",
    "encounterType system": "http://smartplatforms.org/terms/codes/EncounterType#",
    "encounterType identifier": "ambulatory"
  },
  "bp position title": "Sitting",
  "bp position identifier": "33586001",
  "bp position system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
  "bp site title": "Right arm",
  "bp site identifier": "368209003",
  "bp site system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
  "bp method title": "Auscultation",
  "bp method identifier": "ausculation",
  "bp method system": "http://smartplatforms.org/terms/codes/BloodPressureMethod#",
  "bp diastolic unit": "mm[Hg]",
  "bp diastolic value": 82,
  "bp diastolic name title": "Intravascular diastolic",
  "bp diastolic name identifier": "8462-4",
  "bp diastolic name system": "http://purl.bioontology.org/ontology/LNC/",
  "bp systolic unit": "mm[Hg]",
  "bp systolic value": 132,
  "bp systolic name title": "Intravascular systolic",
  "bp systolic name identifier": "8480-6",
  "bp systolic name system": "http://purl.bioontology.org/ontology/LNC/",
  "bmi unit": "kg/m2",
  "bmi value": 21.8,
  "bmi name title": "Body mass index",
  "bmi name system": "http://purl.bioontology.org/ontology/LNC/",
  "bmi name identifier": "39156-5"
}
```
"heart_rate_unit": "(beats)/min",
"heart_rate_value": 70,
"heart_rate_name_title": "Heart rate",
"heart_rate_name_system": "http://purl.bioontology.org/ontology/LNC/",
"heart_rate_name_identifier": "8867-4",

"height_unit": "m",
"height_value": 1.8,
"height_name_title": "Body height",
"height_name_system": "http://purl.bioontology.org/ontology/LNC/",
"height_name_identifier": "8302-2",

"oxygen_saturation_unit": "%(HemoglobinSaturation)",
"oxygen_saturation_value": 99,
"oxygen_saturation_name_title": "Oxygen saturation",
"oxygen_saturation_name_system": "http://purl.bioontology.org/ontology/LNC/",
"oxygen_saturation_name_identifier": "2710-2",

"respiratory_rate_unit": "(breaths)/min",
"respiratory_rate_value": 16,
"respiratory_rate_name_title": "Respiration rate",
"respiratory_rate_name_system": "http://purl.bioontology.org/ontology/LNC/",
"respiratory_rate_name_identifier": "9279-1",

"temperature_unit": "Cel",
"temperature_value": 37,
"temperature_name_title": "Body temperature",
"temperature_name_system": "http://purl.bioontology.org/ontology/LNC/",
"temperature_name_identifier": "8310-5",

"weight_unit": "kg",
"weight_value": 70.8,
"weight_name_title": "Body weight",
"weight_name_system": "http://purl.bioontology.org/ontology/LNC/",
"weight_name_identifier": "3141-9"
}

As SDMX:

<Models>
  <Model name="VitalSigns">
    <Field name="date">2009-05-16T12:00:00Z</Field>
    <Field name="encounter">
      <Field name="startDate">2009-05-16T12:00:00Z</Field>
      <Field name="endDate">2009-05-16T16:00:00Z</Field>
      <Field name="facility_name">Wonder Hospital</Field>
      <Field name="facility_adr_country">Australia</Field>
      <Field name="facility_adr_city">WonderCity</Field>
      <Field name="facility_adr_postalcode">5555</Field>
      <Field name="facility_adr_street">111 Lake Drive</Field>
      <Field name="provider_dea_number">325555555</Field>
      <Field name="provider_npi_number">5235235</Field>
      <Field name="provider_email">joshua.mandel@fake.emailserver.com</Field>
      <Field name="provider_name_given">Josuha</Field>
      <Field name="provider_name_family">Mandel</Field>
      <Field name="provider_tel_1_type">w</Field>
      <Field name="provider_tel_1_number">1-235-947-3452</Field>
  </Field>
</Model>

5.13. Indivo Data Models
<Field name="provider_tel_1_preferred_p">true</Field>
<Field name="encounterType_title">Ambulatory encounter</Field>
<Field name="encounterType_system">http://smartplatforms.org/terms/codes/EncounterType#</Field>
<Field name="encounterType_identifier">ambulatory</Field>
</Model>
</Field>
<Field name="bp_position_title">Sitting</Field>
<Field name="bp_position_identifier">33586001</Field>
<Field name="bp_position_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
<Field name="bp_site_title">Right arm</Field>
<Field name="bp_site_identifier">368209003</Field>
<Field name="bp_site_system">http://purl.bioontology.org/ontology/SNOMEDCT/</Field>
<Field name="bp_method_title">Auscultation</Field>
<Field name="bp_method_identifier">auscultation</Field>
<Field name="bp_method_system">http://smartplatforms.org/terms/codes/BloodPressureMethod#</Field>
<Field name="bp_diastolic_unit">mm[Hg]</Field>
<Field name="bp_diastolic_value">82</Field>
<Field name="bp_diastolic_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="bp_diastolic_name_title">Intravascular diastolic</Field>
<Field name="bp_diastolic_name_identifier">8462-4</Field>
<Field name="bp_diastolic_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="bp_systolic_unit">mm[Hg]</Field>
<Field name="bp_systolic_value">132</Field>
<Field name="bp_systolic_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="bp_systolic_name_title">Intravascular systolic</Field>
<Field name="bp_systolic_name_identifier">8480-6</Field>
<Field name="bp_systolic_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="bmi_unit">kg/m²</Field>
<Field name="bmi_value">21.8</Field>
<Field name="bmi_name_title">Body mass index</Field>
<Field name="bmi_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="bmi_name_identifier">39156-5</Field>
<Field name="heart_rate_unit">{beats}/min</Field>
<Field name="heart_rate_value">70</Field>
<Field name="heart_rate_name_title">Heart rate</Field>
<Field name="heart_rate_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="heart_rate_name_identifier">8867-4</Field>
<Field name="height_unit">m</Field>
<Field name="height_value">1.8</Field>
<Field name="height_name_title">Body height</Field>
<Field name="height_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="height_name_identifier">8302-2</Field>
<Field name="oxygen_saturation_unit">%{HemoglobinSaturation}</Field>
<Field name="oxygen_saturation_value">99</Field>
<Field name="oxygen_saturation_name_title">Oxygen saturation</Field>
<Field name="oxygen_saturation_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="oxygen_saturation_name_identifier">2710-2</Field>
<Field name="oxygen_saturation_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="oxygen_saturation_name_identifier">2710-2</Field>
<Field name="respiratory_rate_unit">{breaths}/min</Field>
<Field name="respiratory_rate_value">16</Field>
<Field name="respiratory_rate_name_title">Respiration rate</Field>
<Field name="respiratory_rate_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="respiratory_rate_name_identifier">9279-1</Field>
<Field name="temperature_unit">Cel</Field>
<Field name="temperature_value">37</Field>
<Field name="temperature_name_title">Body temperature</Field>
<Field name="temperature_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="temperature_name_identifier">8310-5</Field>
<Field name="weight_unit">kg</Field>
<Field name="weight_value">70.8</Field>
<Field name="weight_name_title">Body weight</Field>
<Field name="weight_name_system">http://purl.bioontology.org/ontology/LNC/</Field>
<Field name="weight_name_identifier">8310-5</Field>
As a Fact object:

```python
from indivo.models import Encounter, VitalSigns
from indivo.lib.iso8601 import parse_utc_date as date

encounter_fact = Encounter(
    startDate=date("2009-05-16T12:00:00Z"),
    endDate=date("2009-05-16T16:00:00Z"),
    facility_name="Wonder Hospital",
    facility adr country="Australia",
    facility adr city="WonderCity",
    facility adr street="111 Lake Drive",
    provider_dea_number="325555555",
    provider_npi_number="5235235",
    provider_email="joshua.mandel@fake.emailserver.com",
    provider_name given="Josuha",
    provider name family="Mandel",
    provider tel l type="w",
    provider tel l number="1-235-947-3452",
    provider tel l preferred p=True,
    encounterType title="Ambulatory encounter",
    encounterType system="http://smartplatforms.org/terms/codes/EncounterType#",
    encounterType identifier="ambulatory",
)

encounter_fact.save()
```

# NOTE: all vitals readings are OPTIONAL. You don’t need 
# to add all 56 fields here to create a VitalSigns object.

```python
vitals_fact = VitalSigns(
    date=date("2009-05-16T12:00:00Z"),
    encounter=encounter_fact,

    # Blood Pressure
    bp position title="Sitting",
    bp position identifier="33586001",
    bp position system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    bp site title="Right arm",
    bp site identifier="368209003",
    bp site system="http://purl.bioontology.org/ontology/SNOMEDCT/",
    bp method title="Auscultation",
    bp method identifier="auscultation",
    bp method system="http://smartplatforms.org/terms/codes/BloodPressureMethod#",
    bp diastolic unit="mm[Hg]",
    bp diastolic value=82,
    bp diastolic name title="Intravascular diastolic",
    bp diastolic name identifier="8462-4",
    bp diastolic name system="http://purl.bioontology.org/ontology/LNC/",
    bp systolic unit="mm[Hg]",
    bp systolic value=132,
    bp systolic name title="Intravascular systolic",
    bp systolic name identifier="8480-6",
    bp systolic name system="http://purl.bioontology.org/ontology/LNC/",
)```
# Body Mass Index
bmi_unit="kg/m2",
bmi_value=21.8,
bmi_name_title="Body mass index",
bmi_name_system="http://purl.bioontology.org/ontology/LNC/",
bmi_name_identifier="39156-5",

# Heart Rate
heart_rate_unit="(beats)/min",
heart_rate_value=70,
heart_rate_name_title="Heart rate",
heart_rate_name_system="http://purl.bioontology.org/ontology/LNC/",
heart_rate_name_identifier="8867-4",

# Height
height_unit="m",
height_value=1.8,
height_name_title="Body height",
height_name_system="http://purl.bioontology.org/ontology/LNC/",
height_name_identifier="8867-4",

# Oxygen Saturation
oxygen_saturation_unit="%(HemoglobinSaturation)",
oxygen_saturation_value=99,
oxygen_saturation_name_title="Oxygen saturation",
oxygen_saturation_name_system="http://purl.bioontology.org/ontology/LNC/",
oxygen_saturation_name_identifier="2710-2",

# Respiratory Rate
respiratory_rate_unit="(breaths)/min",
respiratory_rate_value=16,
respiratory_rate_name_title="Respiration rate",
respiratory_rate_name_system="http://purl.bioontology.org/ontology/LNC/",
respiratory_rate_name_identifier="9279-1",

# Temperature
temperature_unit="Cel",
temperature_value=37,
temperature_name_title="Body temperature",
temperature_name_system="http://purl.bioontology.org/ontology/LNC/",
temperature_name_identifier="8310-5",

# Weight
weight_unit="kg",
weight_value=70.8,
weight_name_title="Body weight",
weight_name_system="http://purl.bioontology.org/ontology/LNC/",
weight_name_identifier="3141-9",
}

**Indivo Data Model: Simple_Clinical_Note**

**Model Definition**

As SDML:
As a Django Model Class:

```python
from indivo.models import Fact
from django.db import models

class SimpleClinicalNote(Fact):
    date_of_visit = models.DateTimeField()
    finalized_at = models.DateTimeField(null=True)
    visit_type = models.CharField(null=True, max_length=100)
    visit_type_type = models.CharField(max_length=80, null=True)
    visit_type_value = models.CharField(max_length=40, null=True)
    visit_type_abbrev = models.CharField(max_length=20, null=True)
    visit_location = models.CharField(max_length=200, null=True)
    specialty = models.CharField(null=True, max_length=100)
    specialty_type = models.CharField(max_length=80, null=True)
    specialty_value = models.CharField(max_length=40, null=True)
    specialty_abbrev = models.CharField(max_length=20, null=True)
    signed_at = models.DateTimeField(null=True)
    provider_name = models.CharField(null=True, max_length=200)
    provider_institution = models.CharField(max_length=200, null=True)
    chief_complaint = models.CharField(null=True, max_length=255)
    content = models.TextField(null=True)
```

Examples

As SDMJ:

```json
{"__modelname__": "SimpleClinicalNote",
"date_of_visit": "2010-02-02T12:00:00Z",
"finalized_at": "2010-02-03T13:12:00Z",
"visit_type": "Acute Care",
"visit_type_type": "http://codes.indivo.org/visit-types#",
"visit_type_value": "123",
"visit_type_abbrev": "acute",
"visit_location": "Longfellow Medical",
"specialty": "Hematology/Oncology",
"content": "String"}
```
As SDMX:

```xml
<Models>
  <Model name="SimpleClinicalNote">
    <Field name="date_of_visit">2010-02-02T12:00:00Z</Field>
    <Field name="finalized_at">2010-02-03T13:12:00Z</Field>
    <Field name="visit_type">Acute Care</Field>
    <Field name="visit_type_type">http://codes.indivo.org/visit-types#</Field>
    <Field name="visit_type_value">123</Field>
    <Field name="visit_type_abbrev">acute</Field>
    <Field name="visit_location">Longfellow Medical</Field>
    <Field name="specialty">Hematology/Oncology</Field>
    <Field name="specialty_type">http://codes.indivo.org/specialties#</Field>
    <Field name="specialty_value">234</Field>
    <Field name="specialty_abbrev">hem-onc</Field>
    <Field name="signed_at">2010-02-03T13:12:00Z</Field>
    <Field name="provider_name">Kenneth Mandl</Field>
    <Field name="provider_institution">Children's Hospital Boston</Field>
    <Field name="chief_complaint">stomach ache</Field>
    <Field name="content">Patient presents with ...</Field>
  </Model>
</Models>
```

As a Fact object:

```python
from indivo.models import SimpleClinicalNote
from indivo.lib.iso8601 import parse_utc_date as date

simple_clinical_note_fact = SimpleClinicalNote(
    date_of_visit=date("2010-02-02T12:00:00Z"),
    finalized_at=date("2010-02-03T13:12:00Z"),
    visit_type="Acute Care",
    visit_type_type="http://codes.indivo.org/visit-types#",
    visit_type_value="123",
    visit_type_abbrev="acute",
    visit_location="Longfellow Medical",
    specialty="Hematology/Oncology",
    specialty_type="http://codes.indivo.org/specialties#",
    specialty_value="234",
    specialty_abbrev="hem-onc",
    signed_at=date("2010-02-03T13:12:00Z"),
    provider_name="Kenneth Mandl",
    provider_institution="Children's Hospital Boston",
    chief_complaint="stomach ache",
    content="Patient presents with ..."
)
```
5.13.5 Advanced Data-Model Tasks

Adding Advanced Features to a Data-Model

For complicated data models, a simple SDML definition just won’t suffice. For a few specific features, such as *custom object serialization* or *creation-time field validation*, you can define (in python) an extra options file for a data model. This file should be named `extra.py`, and can be dropped into the filesystem next to any data model, as described *below*. The file should contain subclasses of `indivo.data_models.options.DataModelOptions`, each of which describes the options for one data model defined in the `model.py` file in the same directory. Options are:

```python
class indivo.data_models.options.DataModelOptions(\Type)
    Defines optional extra functionality for Indivo datamodels.

    To add options to a datamodel, subclass this class and override its attributes.
```

Currently available options are:

- **model_class_name**: *Required*. The name of the datamodel class to attach to.
- **serializers**: Custom serializers for the data model. Should be set to a subclass of `indivo.serializers.DataModelSerializers`.
- **field_validators**: Custom validators for fields on the data model. A dictionary, where keys are field names on the model, and values are lists of Django Validators to be run against the field.

For example, here’s our options file for the Problem data model:

```python
from indivo.serializers import DataModelSerializers
from indivo.data_models.options import DataModelOptions
from indivo.validators import ExactValueValidator

SNOMED_URI = 'http://purl.bioontology.org/ontology/SNOMEDCT/'

class ProblemSerializers(DataModelSerializers):
    def to_rdf(queryset, result_count, record=None, carenet=None):
        # ... our SMART RDF serializer implementation here ... #
        return 'some RDF'

class ProblemOptions(DataModelOptions):
    model_class_name = 'Problem'
    serializers = ProblemSerializers
    field_validators = {
        'name_system': [ExactValueValidator(SNOMED_URI)],
    }
```

Make sure to restart Indivo for your changes to take effect after you add your `extra.py` file—but there’s no need to *reset* Indivo.

Adding Custom Serializers to a Data-Model

By default, when returning data via the *generic reporting API*, Indivo will attempt to serialize data as *SDMJ* or *SDMX*, depending on the requested response format. If you need your data to come back in other formats, or if the default serializers aren’t smart enough to represent your data model correctly, you can implement custom serializers for the data model.
Indivo X Documentation, Release 2.0.0

Defining the Serializers

Serializers for a data model are implemented as simple methods that take a Django queryset object, and return a serialized string.
For a given data-model, you should define a subclass of
indivo.serializers.DataModelSerializers, and add your desired serializers as methods on the class.
Currently, available serializers are:
to_xml(queryset, result_count, record=None, carenet=None)
returns an XML string representing the model objects in queryset.
Parameters
• queryset (QuerySet) – the objects to serialize
• result_count (integer) – the total number of items in queryset
• record (Record) – the patient record that the objects belong to, if available.
• carenet (Carenet) – the Carenet via which the objects have been retrieved, if available.
Return type string
to_json(queryset, result_count, record=None, carenet=None)
returns a JSON string representing the model objects in queryset.
Parameters
• queryset (QuerySet) – the objects to serialize
• result_count (integer) – the total number of items in queryset
• record (Record) – the patient record that the objects belong to, if available.
• carenet (Carenet) – the Carenet via which the objects have been retrieved, if available.
Return type string
to_rdf(queryset, result_count, record=None, carenet=None)
returns an RDF/XML string representing the model objects in queryset.
Parameters
• queryset (QuerySet) – the objects to serialize
• result_count (integer) – the total number of items in queryset
• record (Record) – the patient record that the objects belong to, if available.
• carenet (Carenet) – the Carenet via which the objects have been retrieved, if available.
Return type string
For example, here’s a (non-functional) implementation of the serializers for the Problems data-model:
from indivo.serializers import DataModelSerializers
class ProblemSerializers(DataModelSerializers):
def to_xml(queryset, result_count, record=None, carenet=None):
return ’’’<Problems>...bunch of problems here...</Problems>’’’
def to_json(queryset, result_count, record=None, carenet=None):
return ’’’[{"Problem": "data here"}, {"Problem": "More data here..."}]’’’

def to_rdf(queryset, result_count, record=None, carenet=None):
return ’’’<rdf:RDF><rdf:Description rdf:type=’indivo:Problem’>...RDF data here...</rdf:Descri

526

Chapter 5. Indivo Hackers


A couple things to note:

- The `to_*()` methods **DO NOT** take `self` as their first argument. Under the hood, we actually rip the methods out of the serializers class and attach them directly to the data-model class.

- The `model_class_name` attribute is required, and indicates which data-model the serializers should be attached to.

### Libraries for Serialization

When serializing models, the following libraries can come in handy:

- **lxml.etree**: Our favorite XML manipulation library. See [http://lxml.de/tutorial.html](http://lxml.de/tutorial.html) for the details. Lxml is required for a running Indivo instance, so it will always be available for import (from `lxml` import `etree`).


### Attaching the Serializers to a Data Model

Adding custom serializers to a data-model is simple: simply set your `DataModelSerializers` subclass to the `serializers` attribute of a `DataModelOptions` subclass in an `extra.py` file (see above for info on adding advanced data-model options).

### Adding Field Validation to a Data-Model

By default, data models defined in SDML are very permissive: all fields are nullable, and there are no constraints on valid data points other than their type (string, date, etc.). In some cases, a data element could satisfy these constraints, but still be invalid. For example, an Indivo Problem must have its name coded using SNOMED, so a problem without a snomed code is invalid.

#### Defining the Validators

In such cases, you can attach validators to the data model. **Django Validators** are essentially just python callables that raise a `django.core.exceptions.ValidationError` if they are called on an invalid data point. We’ve defined a couple of useful validators, though you could use any function you’d like.

For example, here’s a validator that will accept only the value 2:

```python
from django.core.exceptions import ValidationError

def validate_2(value):
    if value != 2:
        raise ValidationError("Invalid value: %s. Expected 2"%str(value))
```

## 5.13. Indivo Data Models
Built in Validators

Django provides a number of built-in validators, for which a full reference exists here: https://docs.djangoproject.com/en/1.2/ref/validators/#built-in-validators.

In addition, Indivo defines a few useful validators in `indivo.validators`:

```python
class indivo.validators.ValueInSetValidator(valid_values, nullable=False)
    Validates that a value is within a set of possible values.
    The optional ‘nullable’ flag determines whether or not the value may also be empty.

class indivo.validators.ExactValueValidator(valid_value, nullable=False)
    Validates that a value is exactly equal to a certain value.
    The optional ‘nullable’ flag determines whether or not the value may also be empty.
```

Attaching Validators to a Data Model

Adding custom validators to a data-model is simple: simply add the validator to the field_validators attribute of a `DataModelOptions` subclass in an `extra.py` file (see above for info on adding advanced data-model options).

For example, let’s add the requirement that Problem names must be coded as snomed. We can write the validator using the built-in `ExactValueValidator`:

```python
from indivo.validators import ExactValueValidator
SNOMED_URI = 'http://purl.bioontology.org/ontology/SNOMEDCT/
snomed_validator = ExactValueValidator(SNOMED_URI)
```

We can then attach it to the `name_system` field of a Problem, which will guarantee that we only accept problems which identify themselves as having a snomed code for their names:

```python
class ProblemOptions(DataModelOptions):
    model_class_name = 'Problem'
    field_validators = {
        'name_system': [snomed_validator]
    }
```

Note that we put `snomed_validator` in a list, since we might theoretically add additional validators to the `name_system` field.

Adding Custom Data-Models to Indivo

As of version 1.1 of Indivo X, we’ve added a feature that makes it much easier to add (in a drag-and-drop fashion) new supported data models to an instance of Indivo. Adding a new data model to Indivo involves:

- Creating the data model definition
- Dropping the data model into the filesystem
- Migrating the database tables to support the new model

Defining the Data Model

As you saw above, data models can be defined in two formats: SDML or Django model classes. Simply produce a definition in one of the two forms, and save it to a file named `model.sdml` or `model.py`. 
Dropping the Definition into the Filesystem

Indivo data models currently have the following layout on the filesystem:

```
indivo_server/
    indivo/
        ...
        data_models/
            core/
                allergy/
                    model.[sdml | py]
                    example.[sdmj | sdmx | py]
                    extra.py
            ...
        contrib/
```

The `indivo/data_models/core/` directory contains all of our built-in data models, and you shouldn’t modify it. Since you are ‘contributing’ a data model to Indivo, add your data model to the `indivo/data_models/contrib/` directory. Simply:

- **Create a new subdirectory** under `indivo/data_models/contrib/`.
- **Drop your model definition** into that directory. This file **MUST BE NAMED MODEL.PY OR MODEL.SDML** to be identified as a data model.
- **Add (optional) example files** into that directory. Files should be named `example.sdmj`, `example.sdmx`, or `example.py`, and should be example instances of the data model as *SDMJ*, *SDMX*, or *Fact objects* respectively. If present, they will help others use and document your data model.
- **Add an (optional) extras file** to the directory. The file must be named `extra.py`, and may contain extra options for your data-model, such as *custom serializers*.
- **Your final directory structure** should now look something like:

```
indivo_server/
    indivo/
        ...
        data_models/
            core/
                allergy/
                    model.[sdml | py]
                    example.[sdmj | sdmx | py]
                    extra.py
            ...
            contrib/
                your_data_model/
                    model.[sdml | py]
                    example.[sdmj | sdmx | py]
                    extra.py
```

Migrating the Database

Indivo relies on the [South migration tool](http://south.aeracode.org) to get the database synced with the latest data models. Once you’ve dropped your data model into the filesystem, South should be able to detect the necessary changes.

To detect the new model and generate migrations for it, run (from the `indivo_server` directory):

```
python manage.py schemamigration indivo --auto
```
You should see output like:

```
+ Added model indivo.YOURMODELNAME
Created 0018_auto__add_model_YOURMODELNAME.py. You can now apply this migration with: ./manage.py migrate
```

To do a quick sanity check that you aren’t about to blow away your database, run:

```
python manage.py migrate indivo --db-dry-run -v2
```

This should output the SQL that will be run. Make sure this looks reasonable, ESPECIALLY if you are running Indivo on Oracle, where the South tool is still in alpha. If the SQL looks reasonable, go ahead and run the migration, with:

```
python manage.py migrate indivo
```

And you’re all set!

Next Steps

Make sure to restart Indivo for your changes to take effect.

See also:

Now you’ve added a new data model to Indivo: Congratulations! It can be stored in the database and queried via the API.

But until you map a Schema to it, you won’t be able to actually add data to your new model. To learn more, see:

- Data Pipeline
- Adding Custom Schemas to Indivo
- Adding Custom Transforms to Indivo

5.14 Indivo Simple Data Modeling Language (SDML)

5.14.1 Introduction

As of version 1.1, Indivo supports drag-and-drop substitution of data models within the filesystem. We expect that this will encourage Indivo administrators to try their hands at building new data models for use with Indivo, but we recognize that many will not have the python expertise necessary to implement models using the Django web framework. To enable the rapid development and deployment of very simple data models without the overhead of learning Django, we have defined a basic language for representing data models: Simple Data Modeling Language, or SDML.

Data in Indivo is written to the database as python objects using Django’s Object-Relational Mapper: this will be equally problematic for those without python experience. Therefore, we have also defined languages for representing data in JSON or XML with similar degrees of simplicity: Simple Data Modeling JSON (SDMJ) and Simple Data Modeling XML (SDMX).

See also:

This document contains the specifications for these languages. For information on how and when to use them, see:
5.14.2 Defining Data Models: SDML

There are a number of existing languages for defining data models (see http://en.wikipedia.org/wiki/Category:Data_modeling_languages for a listing). However, most of these are much too generalized and powerful for our use case. What we want is a language that is very simple to use (low learning curve), and very limited in functionality (medical data models shouldn’t be too complex, and shouldn’t need most of the capabilities of a relational database). SDML therefore uses JSON’s syntax exactly, but has restrictions that allow us to define Django models with it.

Syntax

• A data model in SDML is represented by a JSON object: 

• Attributes are represented by JSON pairs: ‘name’:‘value’

• All attributes are optional.

• The special attribute named ‘__modelname__’ defines the name of the model.

• Fieldtypes (which are defined by the ‘value’ of an attribute) are restricted to the following:
  – Number
  – String
  – Date
  – CodedValue
  – ValueAndUnit
  – Address
  – Name
  – Telephone
  – Pharmacy
  – Provider
  – VitalSign
  – BloodPressure
  – ValueRange
  – QuantitativeResult
  – One-to-One
  – One-to-Many

• All types but One-to-One and One-to-Many are indicated by a simple string (i.e. ‘Number’ | ‘CodedValue’ | ‘Provider’)

• One-to-One fields are indicated by a sub-object, and may be nested arbitrarily:

```json
{
    "__modelname__": "mymodel",
    "field1": "Date",
    "field2": {
        "__modelname__": "mysubmodel",
        "subfield1": "String",
        "subfield2": "Number",
        "subfield3": {
            "__modelname__": "mysubsubmodel",
            "subsubfield1": "String"
        }
    }
}
```

• One-to-Many fields are indicated by a list containing a definition of a sub-object:

```json
{
    "__modelname__": "mymodel",
    "field1": "Date",
    "field2": [{
        "__modelname__": "mysubmodel",
        "subfield1": "String",
        "subfield2": "Number"
    }]
}
```

And that’s it.

**Example**

Here is an example definition of a medication data model (more complicated than our model, actually) using SDML:

```json
{
    "__modelname__": "TestMedication",
    "name": "String",
    "date_started": "Date",
    "date_stopped": "Date",
    "brand_name": "String",
    "route": "String",
    "prescription": {
        "__modelname__": "TestPrescription",
        "prescribed_by_name": "String",
        "prescribed_by_institution": "String",
        "prescribed_on": "Date",
        "prescribed_stop_on": "Date"
    },
    "fills": [{
        "__modelname__": "TestFill",
        "date_filled": "Date",
        "supply_days": "Number",
        "filled_at_name": "String"
    }]
}
```

This definition will create three new data models: `TestMedication`, `TestPrescription`, and `TestFill`. It will add a one-to-one relation between TestMedication and TestPrescription, and a one-to-many relation between TestMedication and TestFill. That is to say, each TestMedication might have one prescription and multiple fills.
5.14.3 Representing Data: SDMJ and SDMX

Simple Data Modeling JSON (SDMJ) and Simple Data Modeling XML (SDMX) are two nearly identical methods of representing data that matches an SDML definition. The only difference is the form of the envelope around the data.

SDMJ

SDMJ looks exactly like SDML, with four key differences:

- The datatypes in SDML (‘Number’, ‘String’, ‘Date’) are replaced by the datapoints in SDMJ
- Each model has an attribute ‘__documentid__', specifying the ID of the source document
- Since all attributes are optional, any attribute may be omitted in SDMJ.
- In one-to-many attributes, SDMJ actually specifies multiple datapoints, instead of just defining a submodel in a list.

Here’s an example SDMJ document matching the SDML definition above:

```json
{
    "__modelname__": "TestMedication",
    "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
    "name": "ibuprofen",
    "date_started": "2010-10-01T00:00:00Z",
    "date_stopped": "2010-10-31T00:00:00Z",
    "brand_name": "Advil",
    "prescription": {
        "__modelname__": "TestPrescription",
        "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
        "prescribed_by_name": "Kenneth D. Mandl",
        "prescribed_by_institution": "Children’s Hospital Boston",
        "prescribed_on": "2010-09-30T00:00:00Z",
        "prescribed_stop_on": "2010-10-31T00:00:00Z"
    },
    "fills": [
        {
            "__modelname__": "TestFill",
            "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
            "date_filled": "2010-10-01T00:00:00Z",
            "supply_days": "15",
            "filled_at_name": "CVS"
        },
        {
            "__modelname__": "TestFill",
            "__documentid__": "b1d83191-6edd-4aad-be4e-63117cd4c660",
            "date_filled": "2010-10-16T00:00:00Z",
            "supply_days": "15",
            "filled_at_name": "CVS"
        }
    ]
}
```

Note: we’ve removed the ‘route’ attribute as it was not required, and have added two fills. This will result in 4 Fact objects being saved to the database: one TestMedication, one TestPrescription, and two TestFills.
SDMX

SDMX looks exactly like SDMJ, with the exceptions that:

• It’s XML
• attribute-value pairs are represented as `<Field name="attribute_name">attribute_value</Field>`
• The `__modelname__` attribute is pulled out as a toplevel tag with `documentId` as an attribute: `<Model name="model_name" documentId="id"`
• Each `<Model />` tag contains an attribute `documentId`, specifying the ID of the source document
• In order to represent multiple toplevel datapoints, SDMX must always live under a root `<Models>` tag.

Here’s the same example document we just saw as SDMJ in SDMX form:

```xml
<Models>
  <Model name="TestMedication" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
    <Field name="date_started">2010-10-01T00:00:00Z</Field>
    <Field name="name">ibuprofen</Field>
    <Field name="brand_name">Advil</Field>
    <Field name="date_stopped">2010-10-31T00:00:00Z</Field>
    <Field name="prescription">
      <Model name="TestPrescription" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
        <Field name="prescribed_by_name">Kenneth D. Mandl</Field>
        <Field name="prescribed_by_institution">Children’s Hospital Boston</Field>
        <Field name="prescribed_on">2010-09-30T00:00:00Z</Field>
        <Field name="prescribed_stop_on">2010-10-31T00:00:00Z</Field>
      </Model>
    </Field>
    <Field name="fills">
      <Models>
        <Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
          <Field name="date_filled">2010-10-01T00:00:00Z</Field>
          <Field name="supply_days">15</Field>
          <Field name="filled_at_name">CVS</Field>
        </Model>
        <Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
          <Field name="date_filled">2010-10-16T00:00:00Z</Field>
          <Field name="supply_days">15</Field>
          <Field name="filled_at_name">CVS</Field>
        </Model>
      </Models>
    </Field>
  </Model>
</Models>
```

For those who like to work with XML in conjunction with schemas, here’s an XSD which describes SDMX and can be used to validate it:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:complexType name="ModelType">
    <xs:sequence>
      <xs:element name="Field" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType mixed="true">
          <xs:choice>
            <xs:element name="Model" type="ModelType" minOccurs="0" maxOccurs="1"/>
            <xs:element name="Models" type="ModelsType" minOccurs="0" maxOccurs="1"/>
          </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

534 Chapter 5. Indivo Hackers
Representing Dates

Strings and Numbers in SDMX and SDMJ should simply be input as JSON or XML literals as appropriate. Dates work the same way, with the exception that they need to be formatted (as with all dates and date-times in Indivo) as ISO 8601 UTC timestamps as described in Indivo Basic Data Formats.

5.15 Data Pipeline

5.15.1 Introduction

The Indivo Data Pipeline is the set of processes that take input data as it enters Indivo, extract and store clinical datapoints (Facts), and make those datapoints available as output via the API. As of version 1.1 of Indivo X, every component of the pipeline is substitutable: you can add new formats for input data, new ways to store the data, and new methods to output them.

Let’s start with some vocabulary:

Schema A description of a format for data, which can also be used to validate that data. For example, our SDMX Schema defines the format of incoming data in our SDMX specification language, and can be used to determine whether an XML document is valid SDMX. In Indivo, we use schemas to define the formats in which we accept data via the API, and validate that data as it comes in. Right now, schemas can only take the form of XSDs, since we accept input data only in XML form, but in the future this might change. More on Indivo schemas here.

Document A collection of data, formatted according to a Schema. When you use our API to add data to Indivo, the data you send to Indivo constitutes one document.

Transform A tool that takes as input some data which validates against a schema, and outputs it in a format consistent with Indivo’s Data Models. We currently accept two types of transforms: XSLTs and Python code. We also understand several output formats for the data after it has been transformed. More on transforms in Indivo here.

Data Model A definition of the format in which processed data is stored in Indivo’s database. Each ‘data model’ corresponds to a type of clinical information. For example, our Medication Data Model describes how we represent a processed medication in the database. We currently accept two types of data model definitions. More on Indivo Data Models here.
Fact  A single datapoint: an instance of a data model. For example, one Medication is a Fact, as is one Problem, etc. The medication fact is an object whose format is defined by the Medication Data Model. When you take input data in the form of an XML Document and run it through a Transform, you arrive at a Fact object that can be stored in the database.

Serialization  The process of taking a Fact object and converting it to an output format, suitable for returning in response to an API call. For example, our current Reporting API outputs Fact objects serialized to XML or JSON.

5.15.2 The Pipeline

As you may have surmised from the above definitions, the data pipeline in Indivo is actually pretty simple, and consists of five steps:

1. Identification. An incoming document is examined, and its type determined (right now, since data comes only in XML documents, a document’s type can be uniquely constructed from it’s root nodename and namespace, i.e. http://indivo.org/vocab/xml/documents#Allergy).

2. Validation. The identified document is matched against its schema. If it is invalid, the pipeline terminates, and an error is thrown.

3. Transformation. The validated document is then processed using its transform. If the output matches one of the valid transform output formats, it is converted into one or more Fact objects, ready for storage.

4. Storage. The processed Facts are written to the database.

5. Retrieval. When an app makes an API call using the Reporting API, the database is queried for matching Facts. Those facts are then serialized into the required output format (i.e., XML or JSON) and returned to the app.

With the new data pipeline, Schemas, Transforms, and Data Models are all substitutable: you can add and replace them at will.

Notice that there is not a one-to-one relationship between incoming documents and processed fact objects. This allows for Indivo to accept schemas like a CCR, which contains many facts. A parsed CCR document might end up outputting many Problem, Medication, Allergy, Lab, or other facts, even though there was only one input document.

Conversely, there is not one document type associated with producing one type of fact object. This allows Indivo to accept the same type of data in many formats. For example, you could get a medication fact from our standard medication document, but you could also get a medication fact from a CCR.

The data pipeline is activated whenever new data is added to Indivo using document creation API calls, and whenever data is retrieved from Indivo using the reporting API calls. The following calls add new documents to Indivo, and therefore feed data into the pipeline:

- POST /records/{RECORD_ID}/documents/
- PUT /records/{RECORD_ID}/documents/external/{APP_ID}/{EXTERNAL_ID}
- POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace
- PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace/external/{APP_ID}/{EXTERNAL_ID}
- POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/
- PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL_TYPE}/external/{APP_ID}/{EXTERNAL_ID}
- POST /accounts/{ACCOUNT_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}/accept

All of the reporting calls retrieve collections of processed Fact objects from Indivo, and thus rely on the tail end of the processing pipeline.
Figure 5.1: The Indivo Data Pipeline, Visualized
5.15.3 Learning More

See also:

More information on the components of the data pipeline can be found here:

- Indivo Schemas
- Indivo Transforms
- Indivo Data Models

More information on customizing the data pipeline can be found here:

- Adding Custom Schemas to Indivo
- Adding Custom Transforms to Indivo
- Adding Custom Data-Models to Indivo

5.16 Indivo Experimental APIs

The Indivo experimental APIs are used to evaluate new potential features. They are not guaranteed to be supported, consistent, or used in production. When an API is experimental, it is under the /experimental/ URL base. An API graduates to full support by being moved into the normal tree of API URLs, which requires a change in all clients that implement it. This is done on purpose: to ensure that experimental APIs cannot be depended upon, ever, until they graduate to full support.

5.16.1 PubSub

Status: first design, no implementation

Applications may want to be notified of changes that occur in the record, so they can take action without having to constantly poll the Indivo server. We call these subscriptions.

Add a subscription

POST /experimental/records/{record_id}/apps/{pha_email}/subscriptions/
url={url}&
callback={callback_url}

<Subscription id="..." />

A subscription is created by an app on a specific URL, within a record. The URL is relative to the record, i.e. /reports/minimal/medications/. For now only report URLs are supported.

The callback URL is the URL that Indivo should connect to when an update pertaining to that subscription occurs.

View subscriptions
GET /experimental/records/{record_id}/apps/{pha_email}/subscriptions/

<Subscriptions>
  <Subscription id="..">
    <expires_at>...</expires_at>
    <url>...</url>
  </Subscription>
</Subscriptions>

A subscription lasts 3 months by default, and then automatically expires and must be renewed.

Remove subscription

DELETE /experimental/records/{record_id}/apps/{pha_email}/subscriptions/{subscription_id}

Behavior of a subscription ping

When an app is to be notified of a change, if, for example, that app subscribes to medications, then a new medication would trigger:

POST <callback_url>
record_id={record_id}&
url={url}

Pings based on a subscription are expect a 200 successful HTTP code. If they receive a 500 or other error code, the ping will be tried again with exponential backoff, using 1 minute as initial retry interval, and a factor of 2 between retries.

5.17 Generic Reports

5.17.1 Overview

To compliment the introduction of pluggable data models, Indivo provides the ability to run ‘generic’ reports over all data models. These reports support the API Query Interface, and provide an out of the box solution for reporting over core and contributed data models, with the possibility for customization.

5.17.2 API Calls

- GET /records/{RECORD_ID}/reports/{DATA_MODEL}/
- GET /carenets/{CARENET_ID}/reports/{DATA_MODEL}/

5.17.3 Response Formats

Generic reports provide two default response formats, and the ability to implement custom implementations for each. Formats are specified through a response_format query parameter, and the response’s Internet media type will match the requested value.
5.17.4 Customization

The selection of items for return in generic reports can already be customized by taking advantage of the functionality made available through the Query API. In order to customize the format in which those items are returned, you can implement custom serializers for individual data models.

5.17.5 Limitations

The provided processing mechanisms for JSON and XML response formats have protections against recursion, which might produce undesirable output for certain use cases.

- If data models have circular or repeated references, all repeated references after the first will be excluded.

5.18 Indivo Localization

Indivo is developed entirely in America, and all of its text is written in English. However, the value of a PCHR should not be limited to those who can read a specific language, and all of Indivo’s core functionality is not language-specific. Indivo therefore provides functionality for Localization (abbreviated L10n), or adapting the software for use in a specific region by translating its text. Indivo builds upon Django’s native support for L10n in order to accomplish this.

5.18.1 Overview

There are four main steps involved in localizing Indivo: marking strings for translation, creating language files for a specific locale, translating the strings, and compiling the translated strings. This document will cover each of these steps. Note that all Indivo Localization is implemented in our reference UI Server, as Indivo Server itself produces pure XML, not human-readable text. Any Indivo UI app written in Django could use the same steps to achieve localization, however.

5.18.2 Supported Locales

You may not have to go to the trouble to do any translation. Below is a list of currently supported languages that Indivo has already been translated into:

- en_US: American English

If you found your language on the above list, simply edit your settings.py file as described below and Indivo will be automatically translated for you.
5.18.3 Marking Strings for Translation

In order to translate Indivo, all human-readable text produced by the UI must be identified for translation. We’ve taken a first stab and identified most of the strings that we think need to be translated, but we may well have missed some. If you find an untranslated string in the source code that needs translation, you can mark it as follows:

- Surround the text with `{% trans %}` tags. For example,

```html
<h>ENGLISH HEADER</h>
```

becomes

```html
<h>{% trans "ENGLISH HEADER" %}</h>
```

- If the text is a block paragraph, or contains other Django template tags, use the `{% blocktrans %}` tag:

```html
<p>Now I’m talking about something important! And refering to the variable {{var}}!</p>
```

becomes

```html
<p>{% blocktrans %}Now I’m talking about something important! And refering to the variable {{var}}!{% endblocktrans %}</p>
```

- Note: Indivo UI Server produces text in `.html` files, `.ejs` files, and `.js` files. We’ve added some code to allow you to treat all of these files the same way: simply wrap the string you want translated in the appropriate Django tags.

- For more detailed information, see Django’s documentation on this here.

- Tell us about your changes! Submit a github pull request, and we will incorporate your tagged strings so that others can translate them into all of the Indivo supported locales.

5.18.4 Creating Message Files

Generate the File

Once you’ve indicated which strings should be translated, Django will create a single file containing all of the strings you need to translate in order to localize Indivo. This file, called a ‘message file’, has the extension `.po` and can be created by running (from the top-level `indivo_ui_server` directory):

```
python manage.py makemessages -l LOCALE -e .html,.ejs,.py
```

Where `LOCALE` refers to the language code you wish to translate to, indicated in what Django calls ‘locale name format’. Basically, this means that the language code takes the form `ll[_CC]`, where the first two characters, lowercase, represent the desired language, and the optional last two characters, uppercase, represent the country variant of the language. For example, `it` for italian, and `pt_BR` for Brazilian portuguese. For a complete listing of the locales that Django current supports, see the listing in their source code.

A Look at the File

If you open your `*.po` file, which was placed in `indivo_ui_server/locale/LOCALE/LC_MESSAGES/`, you will see, for each marked string, the following structure:

```
#: path/to/python/module.py:23
msgid "Welcome to my site."
msgstr ""
```
The line beginning with # indicates the source file where the message came from, the msgid line is the original English text to be translated, and the msgstr line is where you will write your translations.

5.18.5 Translating Strings

Unfortunately, Django isn’t smart enough to actually do all of the translation for you (and if it did, Indivo would come out looking like it had been run through Google translate). To get an accurate translation of Indivo into your local language, you will need to go through the .po file generated in the previous section and translate each string indicated by a line beginning with msgid. For each such string, translate the string into your local language, and place the result in the line just below it (starting with msgstr). This is the tedious part of localization, but it is a necessary one to insure a complete translation.

5.18.6 Compiling Message Files

Now that you’ve translated Indivo, you need to get those translations into a pre-compiled format so that Django can be efficient in translating Indivo in realtime. This is done with a simple command:

```
python manage.py compilemessages
```

This compiles your *.po file into a *.mo file, which Django uses when Indivo is running.

**Note:** If you ever go back and change any of your translated strings in the *.po file, you’ll need to recompile the file into a *.mo file. Likewise, if you mark additional strings for translation, you’ll need to regenerate the message file for the new strings to be available for translation.

5.18.7 Wrapping Up

And that’s it! You’ve successfully translated Indivo. A few final things to consider:

- If you ever go back and change any of your translated strings in the *.po file, you’ll need to recompile the file into a *.mo file. Likewise, if you mark additional strings for translation, you’ll need to regenerate the message file for the new strings to be available for translation.

- You still need tell Indivo to translate into your particular language: you can do this by editing `indivo_ui_server/settings.py` and changing the LANGUAGE_CODE setting to match your current locale. Note that in `settings.py`, the LANGUAGE_CODE should be of the form `ll[-cc]`, not `ll[CC]` (note the dash separator and lowercase country code).

- If you’ve translated Indivo into a new language, we want to hear about it! Submit your translation to us as a pull request on github, and we’ll make sure that others have access to it.

- For a more detailed description of how Django handles L10n, see https://docs.djangoproject.com/en/1.2/topics/i18n/localization/.

5.19 Messaging and Notifications

In Indivo X, apps and users can message or notify each other. This page describes the various means of communication.

5.19.1 Messaging

Messaging in Indivo is very much like email: every message has a subject and a body.
Records vs. Accounts and Routing

Are messages sent to a record, or to an account? In Indivo X, messages can be sent to either one, but they always end up in an account’s inbox (records don’t have inboxes). In other words, if an app sends a message to a record, it is routed to a set of accounts with a note indicating which record it pertains to. The rules for routing record messages are, for now simple: the owner of the record and any account that has full access to that record receive a copy of a message sent to that record.

Message Content

Messages have a subject, a body, and a body type, which indicates the formatting of the body. (The subject can only be plain text.) The two body types supported at this time are plaintext and markdown. Plaintext is what it sounds like: text that is shown as is, without any HTML interpretation. Markdown is a format that allows for simple markup features like bold, italics, and links, without all the complexity (and security complications) of HTML.

Specifics of Markdown

Markdown support in Indivo is in so-called “safe mode,” where any HTML markup is silently stripped from the markdown content, and only markdown syntax is transformed into HTML bold, italics, and simple links. All links open up in new browser windows, except for links that are meant to send the user to the app that sent the message.

Specifically, with the following markdown syntax:

Go [back to the FDA app]({APP_BASE}/message?id={MESSAGE_ID}) for more information.

Notice the URL contains standard URL template variable notation, using curly brackets. The variables {APP_BASE} and {MESSAGE_ID} are substituted appropriately, and because this is a link to an app, clicking it will not open a new window, but rather send the user to that app, with the given URL.

Messaging a Record

Messaging an Account

5.19.2 Notifications

Notifications only pertain to records.

5.20 Python Client Library Reference

```python
class IndivoClient (server_params, consumer_params, resource_token=None, **state_vars)
```

The Indivo python client. The client should be initialized with the following arguments:

• `server_params`: A dictionary containing information about the API server. The dictionary should have two keys: `api_base`, the server location from which the API is served, and `authorization_base`, the server location to which the user should be redirected to perform OAuth authorization.

• `consumer_params`: A dictionary containing information about your app. The dictionary should have two keys: `consumer_key`, the OAuth consumer key for your app, and `consumer_secret`, the OAuth consumer secret for your app.

• `resource_token`: **Optional.** A token (request, access, or session) with which to sign requests. The token should be a dictionary with two keys: `oauth_token` and `oauth_token_secret`.  

5.20. Python Client Library Reference
• **state_vars**: Optional. Additional state to track with the API. This will be used to fill in url parameters when available. For example, if you create an IndivoClient with `pha_email='myapp@apps.indivo.org'`, then making an API call like `GET /apps/{PHA_EMAIL}` will automatically fill in the url, without you needing to pass the `pha_email` argument into the call. This is useful with variables that are unlikely to change within the use of a single client object, such as `pha_email` or `record_id`. You can override the variables in individual API calls if need be, however.

```python
IndivoClient.get(uri, body={}, headers={}, **uri_params)
```
Make a signed HTTP GET request against Indivo. Arguments are:

• **uri**: The URI against which to make the request. Optionally, the URI may have templatatable parameters, which should take the form of `{VAR_NAME}`. Such variables must be passed in via the `uri_params` argument or be present as a state variable on the `IndivoClient` instance itself, or a `KeyError` will be raised.

• **body**: Optional. A dictionary containing querystring parameters to add to the request, for example: `{ 'record_id': 'abcde' }`.

• **headers**: Optional. A dictionary containing additional HTTP headers to add to the request.

• **uri_params**: Optional. Additional parameters to be templated into the uri. Parameter names should be the lower-cased equivalent of uri parameters. For example, if the URI contains `{PHA_EMAIL}`, `uri_params` should contain `pha_email='someapp@apps.indivo.org'`.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

```python
IndivoClient.put(uri, body='', headers={}, content_type=None, **uri_params)
```
Make a signed HTTP PUT request against Indivo. Arguments are:

• **uri**: The URI against which to make the request. Optionally, the URI may have templatatable parameters, which should take the form of `{VAR_NAME}`. Such variables must be passed in via the `uri_params` argument or be present as a state variable on the `IndivoClient` instance itself, or a `KeyError` will be raised.

• **body**: Optional. The body of the request. It should be either a raw data string, or a dictionary containing form-data parameters.

• **headers**: Optional. A dictionary containing additional HTTP headers to add to the request.

• **content_type**: Optional. The MIME type of the data submitted in the PUT request. Defaults to `application/x-www-form-urlencoded`.

• **uri_params**: Optional. Additional parameters to be templated into the uri. Parameter names should be the lower-cased equivalent of uri parameters. For example, if the URI contains `{PHA_EMAIL}`, `uri_params` should contain `pha_email='someapp@apps.indivo.org'`.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

```python
IndivoClient.post(uri, body='', headers={}, content_type=None, **uri_params)
```
Make a signed HTTP POST request against Indivo. Arguments are:

• **uri**: The URI against which to make the request. Optionally, the URI may have templatatable parameters, which should take the form of `{VAR_NAME}`. Such variables must be passed in via the `uri_params` argument or be present as a state variable on the `IndivoClient` instance itself, or a `KeyError` will be raised.
**body**: Optional. The body of the request. It should be either a raw data string, or a dictionary containing form-data parameters.

**headers**: Optional. A dictionary containing additional HTTP headers to add to the request.

**content_type**: Optional. The MIME type of the data submitted in the POST request. Defaults to application/x-www-form-urlencoded.

**uri_params**: Optional. Additional parameters to be templated into the uri. Parameter names should be the lower-cased equivalent of uri parameters. For example, if the URI contains \{PHA_EMAIL\}, uri_params should contain pha_email='someapp@apps.indivo.org'.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.get_surl_credentials()

Generate a token and secret for signing URLs. This token/secret are based on the client’s current resource token (which should be an access token). SURL credentials are required in order to use a UI Server widget: they delegate access to the UI Server to make API calls on behalf of a user app.

Returns a dictionary with two keys: token and secret.

IndivoClient.account_create([body={}, headers={}, content_type=None])

Create a new account, and send out initialization emails.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_search([body={}, headers={}])

Search for accounts by name or email.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/search for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_info(account_email=None[, body={}, headers={}])

Display information about an account.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.get_connect_credentials(account_email=None, pha_email=None[, body={}, headers={}, content_type=None])

Get OAuth credentials for an app to run in Connect or SMART REST mode.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/connect_credentials for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.delete_user_preferences(account_email=None, pha_email=None[, headers={}])

Delete all app-specific User Preferences for an account.
IndivoClient.get_user_preferences(account_email=None, pha_email=None, body={}, headers={})

Get app-specific User Preferences for an account.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpbin2.Response, and content is the raw content returned by Indivo.

IndivoClient.set_user_preferences(account_email=None, pha_email=None, body={}, headers={}, content_type=None)

Set app-specific User Preferences for an account.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /accounts/{ACCOUNT_EMAIL}/apps/{PHA_EMAIL}/preferences for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpbin2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_authsystem_add(account_email=None, body={}, headers={}, content_type=None)

Add a new method of authentication to an account.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/{ACCOUNT_EMAIL}/authsystems/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpbin2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_password_change(account_email=None, body={}, headers={}, content_type=None)

Change a account’s password.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/{ACCOUNT_EMAIL}/authsystems/password/change for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpbin2.Response, and content is the raw content returned by Indivo.
IndivoClient.account_password_set(account_email=None, body={}, headers={}, content_type=None)

Force the password of an account to a given value.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/(ACCOUNT_EMAIL)/authsystems/password/set for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_username_set(account_email=None, body={}, headers={}, content_type=None)

Force the username of an account to a given value.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/(ACCOUNT_EMAIL)/authsystems/password/set-username for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_check_secrets(account_email=None, primary_secret=None, body={}, headers={})

Validate an account’s primary and secondary secrets.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/(ACCOUNT_EMAIL)/check-secrets/(PRIMARY_SECRET) for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_forgot_password(account_email=None, body={}, headers={}, content_type=None)

Resets an account if the user has forgotten its password.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/(ACCOUNT_EMAIL)/forgot-password for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_inbox(account_email=None, body={}, headers={})

List messages in an account’s inbox.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/(ACCOUNT_EMAIL)/inbox/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.account_send_message(account_email=None, body={}, headers={}, content_type=None)

Send a message to an account.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/inbox/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_inbox_message(account_email=None, message_id=None, body={}, headers={})

Retrieve an individual message from an account’s inbox.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/inbox/{MESSAGE_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_message_archive(account_email=None, message_id=None, body={}, headers={}, content_type=None)

Archive a message.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/inbox/{MESSAGE_ID}/archive for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.account_inbox_message_attachment_accept(attachment_num=None, account_email=None, message_id=None, body={}, headers={}, content_type=None)

Accept a message attachment into the record it corresponds to.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /accounts/{ACCOUNT_EMAIL}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM}/accept for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient\texttt{.account\_info\_set}(\texttt{account\_email=\text{None}[, body=\{\}, \text{headers}=\{\}, \text{content\_type}=\text{None}]})

Set basic information about an account. 

\textit{body} should contain the data to send with the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. \textit{content\_type} should contain the MIME type of the data sent in the \textit{body} argument. See \texttt{POST /accounts/{ACCOUNT\_EMAIL}/info\_set} for valid parameters and the returned content.

This call will return a tuple (\textit{response}, \textit{content}), where \textit{response} is an instance of \texttt{httplib2.Response}, and \textit{content} is the raw content returned by Indivo.

IndivoClient\texttt{.account\_initialize}(\texttt{account\_email=\text{None}[, primary\_secret=\text{None}[, body=\{\}, \text{headers}=\{\}, \text{content\_type}=\text{None}]})

Initialize an account, activating it. 

\textit{body} should contain the data to send with the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. \textit{content\_type} should contain the MIME type of the data sent in the \textit{body} argument. See \texttt{POST /accounts/{ACCOUNT\_EMAIL}/initialize/{PRIMARY\_SECRET}} for valid parameters and the returned content.

This call will return a tuple (\textit{response}, \textit{content}), where \textit{response} is an instance of \texttt{httplib2.Response}, and \textit{content} is the raw content returned by Indivo.

IndivoClient\texttt{.account\_notifications}(\texttt{account\_email=\text{None}[, body=\{\}, \text{headers}=\{\}]})

List an account’s notifications. 

\textit{body} should contain query parameters for the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. See \texttt{GET /accounts/{ACCOUNT\_EMAIL}/notifications/} for valid parameters and the returned content.

This call will return a tuple (\textit{response}, \textit{content}), where \textit{response} is an instance of \texttt{httplib2.Response}, and \textit{content} is the raw content returned by Indivo.

IndivoClient\texttt{.account\_permissions}(\texttt{account\_email=\text{None}[, body=\{\}, \text{headers}=\{\}]})

List the carenets that an account has access to. 

\textit{body} should contain query parameters for the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. See \texttt{GET /accounts/{ACCOUNT\_EMAIL}/permissions/} for valid parameters and the returned content.

This call will return a tuple (\textit{response}, \textit{content}), where \textit{response} is an instance of \texttt{httplib2.Response}, and \textit{content} is the raw content returned by Indivo.

IndivoClient\texttt{.account\_primary\_secret}(\texttt{account\_email=\text{None}[, body=\{\}, \text{headers}=\{\}]})

Display an account’s primary secret. 

\textit{body} should contain query parameters for the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. See \texttt{GET /accounts/{ACCOUNT\_EMAIL}/primary\_secret} for valid parameters and the returned content.

This call will return a tuple (\textit{response}, \textit{content}), where \textit{response} is an instance of \texttt{httplib2.Response}, and \textit{content} is the raw content returned by Indivo.
IndivoClient.record_list(account_email=None, body={}, headers={})

List all available records for an account.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/records/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_reset(account_email=None, body={}, headers={}, content_type=None)

Reset an account to an uninitialized state.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/{ACCOUNT_EMAIL}/reset for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_secret(account_email=None, body={}, headers={})

Return the secondary secret of an account.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /accounts/{ACCOUNT_EMAIL}/secret for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_resend_secret(account_email=None, body={}, headers={}, content_type=None)

Sends an account user their primary secret in case they lost it.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/{ACCOUNT_EMAIL}/secret-resend for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.account_set_state(account_email=None, body={}, headers={}, content_type=None)

Set the state of an account.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /accounts/{ACCOUNT_EMAIL}/set-state for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.all_phas([body={}, headers={}])
List all available userapps.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.all_manifests([body={}, headers={}])
List SMART manifests for all available userapps.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/manifests/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.pha_delete(pha_email=None, headers={})
Delete a userapp from Indivo.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.pha(pha_email=None, body={}, headers={})
Return a description of a single userapp.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.app_document_list(pha_email=None, body={}, headers={})
List app-specific documents.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/{PHA_EMAIL}/documents/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.app_document_create(pha_email=None, body={}, headers={}, content_type=None)
Create an app-specific Indivo document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /apps/{PHA_EMAIL}/documents/ for valid parameters and the returned content.
This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

```
IndivoClient.app_document_create_or_update_ext(external_id=None, pha_email=None, body={}, headers={}, content_type=None)
```

Create an app-specific Indivo document with an associated external id.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

```
IndivoClient.app_document_meta_ext(external_id=None, pha_email=None, body={}, headers={})
```

Fetch the metadata of an app-specific document identified by external id.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

```
IndivoClient.app_document_delete(pha_email=None, document_id=None, headers={})
```

Delete an app-specific document.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

```
IndivoClient.app_specific_document(pha_email=None, document_id=None, body={}, headers={})
```

Retrieve an app-specific document.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

```
IndivoClient.app_document_create_or_update(pha_email=None, document_id=None, body={}, headers={}, content_type=None)
```

Create or Overwrite an app-specific Indivo document.
**body** should contain the data to send with the call. **headers** should contain any additional HTTP headers to pass in with the call. **content_type** should contain the MIME type of the data sent in the **body** argument. See `PUT /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}` for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

---

```python
IndivoClient.app_document_label(pha_email=None, document_id=None[
    body={}, headers={},
    content_type=None]
)
```

Set the label of an app-specific document.

**body** should contain the data to send with the call. **headers** should contain any additional HTTP headers to pass in with the call. **content_type** should contain the MIME type of the data sent in the **body** argument. See `PUT /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/label` for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

---

```python
IndivoClient.app_document_meta(pha_email=None, document_id=None[
    body={}, headers={}
]
)
```

Fetch the metadata of an app-specific document.

**body** should contain query parameters for the call. **headers** should contain any additional HTTP headers to pass in with the call. See `GET /apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/meta` for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

---

```python
IndivoClient.app_manifest(pha_email=None[
    body={}, headers={}
]
)
```

Return a SMART manifest for a single userapp.

**body** should contain query parameters for the call. **headers** should contain any additional HTTP headers to pass in with the call. See `GET /apps/{PHA_EMAIL}/manifest` for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.

---

```python
IndivoClient.app_record_list(pha_email=None[
    body={}, headers={}
]
)
```

Return a list of all records that have this pha enabled.

**body** should contain query parameters for the call. **headers** should contain any additional HTTP headers to pass in with the call. See `GET /apps/{PHA_EMAIL}/records/` for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where `response` is an instance of `httplib2.Response`, and `content` is the raw content returned by Indivo.
IndivoClient.autonomous_access_token(record_id=None, pha_email=None, body={}, headers={}, content_type=None)

Fetch an access token for an autonomous app to access a record.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST `/apps/{PHA_EMAIL}/records/{RECORD_ID}/access_token` for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

IndivoClient.smart_capabilities([body={}, headers={}])

SMART Capabilities

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET `/capabilities/` for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_delete(carenet_id=None, headers={})

Delete a carenet.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE `/carenets/{CARENET_ID}` for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_account_list(carenet_id=None, body={}, headers={})

List the accounts in a carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET `/carenets/{CARENET_ID}/accounts/` for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_account_create(carenet_id=None, body={}, headers={}, content_type=None)

Add an account to a carenet.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST `/carenets/{CARENET_ID}/accounts/` for valid parameters and the returned content.

This call will return a tuple `(response, content)`, where *response* is an instance of `httplib2.Response`, and *content* is the raw content returned by Indivo.
IndivoClient.carenet_account_delete(account_id=None, carenet_id=None[, headers={}])

Remove an account from a carenet.

headers should contain any additional HTTP headers to pass in with the call. See DELETE /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.carenet_account_permissions(account_id=None, carenet_id=None[, body={}, headers={}])

List the permissions of an account within a carenet.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/accounts/{ACCOUNT_ID}/permissions for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.carenet_apps_list(carenet_id=None[, body={}, headers={}])

List Apps within a given carenet.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/apps/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.carenet_apps_delete(pha_email=None, carenet_id=None[, headers={}])

Remove an app from a given carenet.

headers should contain any additional HTTP headers to pass in with the call. See DELETE /carenets/{CARENET_ID}/apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.carenet_apps_create(pha_email=None, carenet_id=None[, body={}, headers={}, content_type=None])

Add an app to a carenet

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /carenets/{CARENET_ID}/apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.carenet_app_permissions(pha_email=None, carenet_id=None[, body={}, headers={}])

Retrieve the permissions for an app within a carenet. NOT IMPLEMENTED.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/apps/{PHA_EMAIL}/permissions for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.read_demographics_carenet(carenet_id=None[, body={}, headers={}])

Read demographics from a carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/demographics for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_document_list(carenet_id=None[, body={}, headers={}])

List documents from a given carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/documents/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_document(document_id=None, carenet_id=None[, body={}, headers={}])

Return a document from a carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_document_meta(document_id=None, carenet_id=None[, body={}, headers={}])

Fetch the metadata of a record-specific document via a carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/documents/{DOCUMENT_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.carenet_record(carenet_id=None, body={}, headers={})

Get basic information about the record to which a carenet belongs.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/record for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_rename(carenet_id=None, body={}, headers={}, content_type=None)

Change a carenet’s name.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /carenets/{CARENET_ID}/rename for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_equipment_list(carenet_id=None, body={}, headers={})

List the equipment data for a given carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/reports/minimal/equipment/ for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_measurement_list(carenet_id=None, lab_code=None, body={}, headers={})

List the measurement data for a given carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/reports/minimal/measurements/{LAB_CODE}/ for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_procedure_list(carenet_id=None, body={}, headers={})

List the procedure data for a given carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/reports/minimal/procedures/ for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.carenet_simple_clinical_notes_list(carenet_id=None[, body={}, headers={}])

List the simple_clinical_notes data for a given carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/reports/minimal/simple-clinical-notes/ for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_generic_list(data_model=None, carenet_id=None[, body={}, headers={}])

List the Model data for a given carenet.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /carenets/{CARENET_ID}/reports/{DATA_MODEL}/ for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.coding_systems_list([body={}, headers={}])

List available coding systems. NOT IMPLEMENTED.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /codes/systems/ for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.coding_system_query(system_short_name=None[, body={}, headers={}])

Query a coding system for a value.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /codes/systems/{SYSTEM_SHORT_NAME}/query for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.exchange_token([body={}, headers={}, content_type=None])

Exchange a request token for a valid access token.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /oauth/access_token for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.request_token_approve(reqtoken_id=None, body={}, headers={}, content_type=None)

Indicate a user’s consent to bind an app to a record or carenet.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /oauth/internal/request_tokens/{REQTOKEN_ID}/approve for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.request_token_claim(reqtoken_id=None, body={}, headers={}, content_type=None)

Claim a request token on behalf of an account.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /oauth/internal/request_tokens/{REQTOKEN_ID}/claim for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.request_token_info(reqtoken_id=None, body={}, headers={})

Get information about a request token.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /oauth/internal/request_tokens/{REQTOKEN_ID}/info for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.session_create([body={}, headers={}, content_type=None])

Authenticate a user and register a web session for them.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /oauth/internal/session_create for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.surl_verify([body={}, headers={}])

Verify a signed URL.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /oauth/internal/surl-verify for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.request_token([body={}, headers={}, content_type=None])
Get a new request token, bound to a record or carenet if desired.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /oauth/request_token for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.smart_ontology([body={}, headers={}])
Fetch the SMART ontology as RDF/XML.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /ontology for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_create([body={}, headers={}, content_type=None])
Create a new record.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /records/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_create_ext(principal_email=None, external_id=None, [body={}, headers={}, content_type=None])
Create a new record with an associated external id.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /records/external/{PRINCIPAL_EMAIL}/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_search([body={}, headers={}])
Search for records by label (usually the same as full name).

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/search for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.record(record_id=None[, body={}, headers={}] )
Get information about an individual record.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.smart_allergies(record_id=None[, body={}, headers={}] )
SMART allergy list, serialized as RDF/XML.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/allergies/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.smart_allergies_instance(record_id=None, model_id=None[, body={}, headers={}] )
Retrieve a specific instance of a SMART Allergy/AllergyExclusion.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/allergies/{MODEL_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_phas(record_id=None[, body={}, headers={}] )
List userapps bound to a given record.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/apps/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.pha_record_delete(record_id=None, pha_email=None[, headers={}] )
Remove a userapp from a record.

headers should contain any additional HTTP headers to pass in with the call. See DELETE /records/{RECORD_ID}/apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient\texttt{.record_pha}(record_id=None, pha_email=None[, body={}, headers={}])

Get information about a given userapp bound to a record.

\textit{body} should contain query parameters for the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where \textit{response} is an instance of httplib2.Response, and \textit{content} is the raw content returned by Indivo.

IndivoClient\texttt{.record_pha_enable}(record_id=None, pha_email=None[, body={}, headers={}, content_type=None])

Enable a userapp for a record.

\textit{body} should contain the data to send with the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. \textit{content_type} should contain the MIME type of the data sent in the \textit{body} argument. See PUT /records/{RECORD_ID}/apps/{PHA_EMAIL} for valid parameters and the returned content.

This call will return a tuple (response, content), where \textit{response} is an instance of httplib2.Response, and \textit{content} is the raw content returned by Indivo.

IndivoClient\texttt{.record_app_document_list}(record_id=None, pha_email=None[, body={}, headers={}])

List record-app-specific documents.

\textit{body} should contain query parameters for the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/ for valid parameters and the returned content.

This call will return a tuple (response, content), where \textit{response} is an instance of httplib2.Response, and \textit{content} is the raw content returned by Indivo.

IndivoClient\texttt{.record_app_document_create}(record_id=None, pha_email=None[, body={}, headers={}, content_type=None])

Create a record-app-specific Indivo document.

\textit{body} should contain the data to send with the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. \textit{content_type} should contain the MIME type of the data sent in the \textit{body} argument. See POST /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/ for valid parameters and the returned content.

This call will return a tuple (response, content), where \textit{response} is an instance of httplib2.Response, and \textit{content} is the raw content returned by Indivo.

IndivoClient\texttt{.record_app_document_create_or_update_ext}(record_id=None, external_id=None, pha_email=None[, body={}, headers={}, content_type=None])

Create or Overwrite a record-app-specific Indivo document with an associated external id.

\textit{body} should contain the data to send with the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. \textit{content_type} should contain the MIME type of the data sent in the \textit{body} argument. See POST
/records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpclient2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_app_document_create_or_update_ext (record_id=None, external_id=None, pha_email=None, body={}, headers={}, content_type=None)

Create or Overwrite a record-app-specific Indivo document with an associated external id.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpclient2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_app_document_meta_ext (record_id=None, external_id=None, pha_email=None, body={}, headers={})

Fetch the metadata of a record-app-specific document identified by external id.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/external/{EXTERNAL_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpclient2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_app_document_delete (record_id=None, pha_email=None, document_id=None, headers={})

Delete a record-app-specific document.

headers should contain any additional HTTP headers to pass in with the call. See DELETE /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpclient2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_app_specific_document (record_id=None, pha_email=None, document_id=None, body={}, headers={})

Retrieve a record-app-specific document.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpclient2.Response, and content is the raw content returned by Indivo.
IndivoClient.record_app_document_label(record_id=None, pha_email=None, document_id=None, body={}, headers={}, content_type=None)

Set the label of a record-app-specific document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/label for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.record_app_document_meta(record_id=None, pha_email=None, document_id=None, body={}, headers={})

Fetch the metadata of a record-app-specific document.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/apps/{PHA_EMAIL}/documents/{DOCUMENT_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.record_pha_setup(record_id=None, pha_email=None, body={}, headers={}, content_type=None)

Bind an app to a record without user authorization.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/apps/{PHA_EMAIL}/setup for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.audit_record_view(record_id=None, body={}, headers={})

Return audits of calls touching *record*.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/audits/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.audit_document_view(record_id=None, document_id=None, body={}, headers={})

Return audits of calls touching *record* and *document_id*. 

5.20. Python Client Library Reference

565
body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/audits/documents/{DOCUMENT_ID}/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

**IndivoClient.audit_function_view**(record_id=None, document_id=None, function_name=None, body={}, headers={})

Return audits of calls to **function_name** touching record and document_id.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/audits/documents/{DOCUMENT_ID}/functions/{FUNCTION_NAME}/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

**IndivoClient.audit_query**(record_id=None, body={}, headers={})

Select Audit Objects via the Query API Interface.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/audits/query/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

**IndivoClient.autoshare_list**(record_id=None, body={}, headers={})

For a single record, list all carenets that a given doctype is autoshared with.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/autoshare/bytype/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

**IndivoClient.autoshare_list_bytype_all**(record_id=None, body={}, headers={})

For a single record, list all doctypes autoshared into carenets.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/autoshare/bytype/all for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.autoshare_create(record_id=None, carenet_id=None, body={}, headers={}, content_type=None)

Automatically share all documents of a certain type into a carenet.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/set for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.autoshare_delete(record_id=None, carenet_id=None, body={}, headers={}, content_type=None)

Remove an autoshare from a carenet.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/autoshare/carenets/{CARENET_ID}/bytype/unset for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_list(record_id=None, body={}, headers={})

List all carenets for a record.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/carenets/ for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_create(record_id=None, body={}, headers={}, content_type=None)

Create a new carenet for a record.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/carenets/ for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.read_demographics(record_id=None, body={}, headers={})

Read demographics from a record.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/demographics for valid parameters and the returned content.

This call will return a tuple *(response, content)*, where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.set_demographics(record_id=None, body={}, headers={}, content_type=None)

Create or update demographics on a record.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/{RECORD_ID}/demographics for valid parameters and the returned content.

This call will return a tuple (*response, content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.documents_delete(record_id=None, headers={})

Delete all documents associated with a record.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /records/{RECORD_ID}/documents/ for valid parameters and the returned content.

This call will return a tuple (*response, content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.record_document_list(record_id=None, body={}, headers={})

List record-specific documents.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/ for valid parameters and the returned content.

This call will return a tuple (*response, content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.document_create(record_id=None, body={}, headers={}, content_type=None)

Create a record-specific Indivo Document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/documents/ for valid parameters and the returned content.

This call will return a tuple (*response, content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.document_create_by_ext_id(record_id=None, external_id=None, pha_email=None, body={}, headers={}, content_type=None)

Create a record-specific Indivo Document with an associated external id.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/{RECORD_ID}/documents/external/{PHA_EMAIL}/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (*response, content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient:\texttt{record\_document\_label\_ext}(\texttt{record\_id=None, external\_id=None, pha\_email=None, body={}, headers={}, content\_type=None})

Set the label of a record-specific document, specified by external id.

\textit{body} should contain the data to send with the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. \textit{content\_type} should contain the MIME type of the data sent in the \textit{body} argument. See \texttt{PUT /records/{RECORD_ID}/documents/external/{PHA_EMAIL}/{EXTERNAL_ID}/label} for valid parameters and the returned content.

This call will return a tuple \texttt{(response, content)}, where \textit{response} is an instance of \texttt{httplib2.Response}, and \textit{content} is the raw content returned by Indivo.

IndivoClient:\texttt{record\_document\_meta\_ext}(\texttt{record\_id=None, external\_id=None, pha\_email=None, body={}, headers={})

Fetch the metadata of a record-specific document identified by external id.

\textit{body} should contain query parameters for the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. See \texttt{GET /records/{RECORD_ID}/documents/external/{PHA_EMAIL}/{EXTERNAL_ID}/meta} for valid parameters and the returned content.

This call will return a tuple \texttt{(response, content)}, where \textit{response} is an instance of \texttt{httplib2.Response}, and \textit{content} is the raw content returned by Indivo.

IndivoClient:\texttt{document\_rels}(\texttt{record\_id=None, document\_id\_1=None, document\_id\_0=None, rel=None, body={}, headers={}, content\_type=None})

Create a new relationship between two existing documents.

\textit{body} should contain the data to send with the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. \textit{content\_type} should contain the MIME type of the data sent in the \textit{body} argument. See \texttt{PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID\_0}/rels/{REL}/{DOCUMENT_ID\_1}} for valid parameters and the returned content.

This call will return a tuple \texttt{(response, content)}, where \textit{response} is an instance of \texttt{httplib2.Response}, and \textit{content} is the raw content returned by Indivo.

IndivoClient:\texttt{record\_specific\_document}(\texttt{record\_id=None, document\_id=None, body={}, headers={})

Retrieve a record-specific document.

\textit{body} should contain query parameters for the call. \textit{headers} should contain any additional HTTP headers to pass in with the call. See \texttt{GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}} for valid parameters and the returned content.

This call will return a tuple \texttt{(response, content)}, where \textit{response} is an instance of \texttt{httplib2.Response}, and \textit{content} is the raw content returned by Indivo.
IndivoClient.document_carenets(record_id=None, document_id=None, body={}, headers={})

List all the carenets into which a document has been shared.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/ for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_document_delete(record_id=None, carenet_id=None, document_id=None, headers={})

Unshare a document from a given carenet.

*headers* should contain any additional HTTP headers to pass in with the call. See DELETE /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.carenet_document_placement(record_id=None, carenet_id=None, document_id=None, body={}, headers={}, content_type=None)

Place a document into a given carenet.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.autoshare_revert(record_id=None, carenet_id=None, document_id=None, body={}, headers={}, content_type=None)

Revert the document-sharing of a document in a carenet to whatever rules are specified by autoshares. NOT IMPLEMENTED.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/carenets/{CARENET_ID}/autoshare-revert for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.record_document_label(record_id=None, document_id=None, body={}, headers={}, content_type=None)

Set the label of a record-specific document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See
PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/label for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpclient2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_document_meta(record_id=None, document_id=None, body={}, headers={})

Fetch the metadata of a record-specific document.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpclient2.Response, and content is the raw content returned by Indivo.

IndivoClient.update_document_meta(record_id=None, document_id=None, body={}, headers={}, content_type=None)

Set metadata fields on a document. NOT IMPLEMENTED.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/meta for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpclient2.Response, and content is the raw content returned by Indivo.

IndivoClient.document_remove_nevershare(record_id=None, document_id=None, headers={})

Remove the nevershare flag from a document. headers should contain any additional HTTP headers to pass in with the call. See DELETE /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpclient2.Response, and content is the raw content returned by Indivo.

IndivoClient.document_set_nevershare(record_id=None, document_id=None, body={}, headers={}, content_type=None)

Flag a document to never be shared, anywhere.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/nevershare for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httpclient2.Response, and content is the raw content returned by Indivo.
IndivoClient.get_documents_by_rel(record_id=None, rel=None, document_id=None, body={}, headers={})

Get all documents related to the passed document_id by a relation of the passed relation-type.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL}/ for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.document_create_by_rel(record_id=None, rel=None, document_id=None, body={}, headers={})

Create a document and relate it to an existing document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL}/ for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.document_create_by_rel_with_ext_id(record_id=None, rel=None, external_id=None, pha_email=None, document_id=None, body={}, headers={}, content_type=None)

Create a document, assign it an external id, and relate it to an existing document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/rels/{REL}/external/{PHA_EMAIL}/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (*response*, *content*), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.
IndivoClient.document_version(record_id=None, document_id=None, body={}, headers={}, content_type=None)

Create a new version of a record-specific document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.document_version_by_ext_id(record_id=None, external_id=None, pha_email=None, document_id=None, body={}, headers={}, content_type=None)

Create a new version of a record-specific document and assign it an external id.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See PUT /records/{RECORD_ID}/documents/{DOCUMENT_ID}/replace/external/{PHA_EMAIL}/{EXTERNAL_ID} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.document_set_status(record_id=None, document_id=None, body={}, headers={}, content_type=None)

Set the status of a record-specific document.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content_type* should contain the MIME type of the data sent in the *body* argument. See POST /records/{RECORD_ID}/documents/{DOCUMENT_ID}/set-status for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.document_status_history(record_id=None, document_id=None, body={}, headers={})

List all changes to a document’s status over time.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/status-history for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of httplib2.Response, and *content* is the raw content returned by Indivo.

IndivoClient.document_versions(record_id=None, document_id=None, body={}, headers={})

Retrieve the versions of a document.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/documents/{DOCUMENT_ID}/versions/ for valid parameters and the returned content.
This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.

```python
IndivoClient.record_send_message(record_id=None, message_id=None, body={}, headers={}, content_type=None)
```

Send a message to a record.

- \(\text{body}\) should contain the data to send with the call. \(\text{headers}\) should contain any additional HTTP headers to pass in with the call. \(\text{content_type}\) should contain the MIME type of the data sent in the \(\text{body}\) argument. See POST /records/{RECORD_ID}/inbox/{MESSAGE_ID} for valid parameters and the returned content.

This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.

```python
IndivoClient.record_message_attach(record_id=None, attachment_num=None, message_id=None, body={}, headers={}, content_type=None)
```

Attach a document to an Indivo message.

- \(\text{body}\) should contain the data to send with the call. \(\text{headers}\) should contain any additional HTTP headers to pass in with the call. \(\text{content_type}\) should contain the MIME type of the data sent in the \(\text{body}\) argument. See POST /records/{RECORD_ID}/inbox/{MESSAGE_ID}/attachments/{ATTACHMENT_NUM} for valid parameters and the returned content.

This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.

```python
IndivoClient.record_notify(record_id=None, body={}, headers={}, content_type=None)
```

Send a notification about a record to all accounts authorized to be notified.

- \(\text{body}\) should contain the data to send with the call. \(\text{headers}\) should contain any additional HTTP headers to pass in with the call. \(\text{content_type}\) should contain the MIME type of the data sent in the \(\text{body}\) argument. See POST /records/{RECORD_ID}/notify for valid parameters and the returned content.

This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.

```python
IndivoClient.record_get_owner(record_id=None, body={}, headers={})
```

Get the owner of a record.

- \(\text{body}\) should contain query parameters for the call. \(\text{headers}\) should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/owner for valid parameters and the returned content.

This call will return a tuple \((\text{response}, \text{content})\), where \(\text{response}\) is an instance of `httplib2.Response`, and \(\text{content}\) is the raw content returned by Indivo.
IndivoClient."record_set_owner"(record_id=None, body={}, headers={}, content_type=None)

Set the owner of a record.

(body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /records/{RECORD_ID}/owner for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient."record_set_owner"(record_id=None, body={}, headers={}, content_type=None)

Set the owner of a record.

(body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See PUT /records/{RECORD_ID}/owner for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient."report_ccr"(record_id=None, body={}, headers={})

Export patient data as a Continuity of Care Record (CCR) document.

(body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/experimental/ccr for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient."equipment_list"(record_id=None, body={}, headers={})

List the equipment data for a given record.

(body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/minimal/equipment/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient."measurement_list"(record_id=None, lab_code=None, body={}, headers={})

List the measurement data for a given record.

(body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/minimal/measurements/{LAB_CODE}/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

5.20. Python Client Library Reference

575
IndivoClient.procedure_list(record_id=None, body={}, headers={})

List the procedure data for a given record.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/minimal/procedures/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.simple_clinical_notes_list(record_id=None, body={}, headers={})

List the simple_clinical_notes data for a given record.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/minimal/simple-clinical-notes/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.generic_list(record_id=None, data_model=None, body={}, headers={})

List the Model data for a given record.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/reports/{DATA_MODEL}/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_shares(record_id=None, body={}, headers={})

List the shares of a record.

body should contain query parameters for the call. headers should contain any additional HTTP headers to pass in with the call. See GET /records/{RECORD_ID}/shares/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.

IndivoClient.record_share_add(record_id=None, body={}, headers={}, content_type=None)

Fully share a record with another account.

body should contain the data to send with the call. headers should contain any additional HTTP headers to pass in with the call. content_type should contain the MIME type of the data sent in the body argument. See POST /records/{RECORD_ID}/shares/ for valid parameters and the returned content.

This call will return a tuple (response, content), where response is an instance of httplib2.Response, and content is the raw content returned by Indivo.
IndivoClient.\texttt{record\_share\_delete}(record\_id=None, other\_account\_id=None[, headers={}])

Undo a full record share with an account.

*headers* should contain any additional HTTP headers to pass in with the call. See \texttt{DELETE /records/{RECORD\_ID}/shares/{OTHER\_ACCOUNT\_ID}} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of \texttt{httplib2.Response}, and *content* is the raw content returned by Indivo.

IndivoClient.\texttt{record\_share\_delete}(record\_id=None, other\_account\_id=None[, body={}, headers={}, content\_type=None])

Undo a full record share with an account.

*body* should contain the data to send with the call. *headers* should contain any additional HTTP headers to pass in with the call. *content\_type* should contain the MIME type of the data sent in the *body* argument. See \texttt{POST /records/{RECORD\_ID}/shares/{OTHER\_ACCOUNT\_ID}/delete} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of \texttt{httplib2.Response}, and *content* is the raw content returned by Indivo.

IndivoClient.\texttt{smart\_generic}(record\_id=None, model\_name=None[, body={}, headers={}])

SMART-compatible alias for the generic list view: returns data models serialized as SMART RDF.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See \texttt{GET /records/{RECORD\_ID}/{MODEL\_NAME}} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of \texttt{httplib2.Response}, and *content* is the raw content returned by Indivo.

IndivoClient.\texttt{smart\_generic\_instance}(record\_id=None, model\_id=None, model\_name=None[, body={}, headers={}])

Retrieve a specific instance of a SMART model.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See \texttt{GET /records/{RECORD\_ID}/{MODEL\_NAME}/{MODEL\_ID}} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of \texttt{httplib2.Response}, and *content* is the raw content returned by Indivo.

IndivoClient.\texttt{get\_version}([body={}, headers={}])

Return the current version of Indivo.

*body* should contain query parameters for the call. *headers* should contain any additional HTTP headers to pass in with the call. See \texttt{GET /version} for valid parameters and the returned content.

This call will return a tuple (response, content), where *response* is an instance of \texttt{httplib2.Response}, and *content* is the raw content returned by Indivo.
5.21 Registering Apps with Indivo

As of version 2.0, Indivo has an official process for adding and managing apps on a given instance. For those of you familiar with using indivo_data.xml for adding apps, the <machine_apps> and <user_apps> tags will no longer be respected, and will not add apps to Indivo.

5.21.1 App Manifests

Previous releases of Indivo have relied on an implicit XML syntax in indivo_data.xml for representing an app within Indivo. Now, applications must provide a declared manifest describing the application and its requirements. We have adopted the SMART Project's syntax for describing apps in manifests, but have added some Indivo-specific parameters to accommodate the way Indivo represents applications. For compatibility, any valid SMART manifest will also be a valid Indivo manifest.

See The SMART Project’s documentation for a description of the basic syntax of a manifest, which is JSON based. Below, we describe only the Indivo-specific modifications.

User Apps

New Fields

An Indivo user-app may define (beyond the SMART-supported manifest fields) any of the following additional properties in its manifest:

- **oauth_callback_url**: A callback URL for Indivo-style oAuth access
- **autonomous_reason**: An explanation for why the app requires offline access to patient records
- **has_ui**: true or false, whether the app can be displayed in a browser.
- **frameable**: true or false, whether the app should be loaded in an iframe in the Indivo UI.
- **indivo_version**: Required version of Indivo for compatibility

Changes from the indivo_data.xml fields

The following fields have changed names to match the SMART manifest fields:

- **email**: Has been renamed *id*, per SMART.
- **start_url_template**: Has been renamed *index*. If using Indivo-style oAuth authentication, the same templating parameters may be passed in the URL (i.e. `{record_id}`)
- **is_autonomous**: Has been moved to the SMART *mode* property. Acceptable modes are:
  - *background*: This app will act like an autonomous Indivo app.
  - *ui*: This app will act like a non-autonomous Indivo app.

Example

```json
{
  "name" : "Problems",
  "description" : "Display a list of problems, or enter new ones."
  "author" : "Arjun Sanyal, Children’s Hospital Boston",
}
"id" : "problems@apps.indivo.org",
"version" : "1.0.0",
"smart_version": "0.4",

"mode" : "ui",
"scope": "record",
"has_ui": true,
"frameable": true,

"icon" : "jmvc/ui/resources/images/app_icons_32/problems.png",
"index": "/apps/problems/start_auth?record_id={record_id}&carenet_id={carenet_id}",
"oauth_callback_url": "/apps/problems/after_auth"
}

UI and Admin Apps

New Fields

An Indivo user-app may define (beyond the SMART-supported manifest fields) any of the following additional properties in its manifest:

• **ui_app**: true or false. Whether the machineapp is a UIApp (‘chrome app’).
• **indivo_version**: Required version of Indivo for compatibility

Changes from the indivo_data.xml fields

The following fields have changed names to match the SMART manifest fields:

• **email**: Has been renamed **id**, per SMART.
• **app_type**: Please use the **ui_app** field, above.

Example

```
{
  "name": "Sample UI App",
  "description" : "The reference Indivo UI App",
  "author" : "Ben Adida, Travers Franckle, Arjun Sanyal, Pascal Pfiffner, Daniel Haas. Children’s Hospital Boston",
  "id" : "chrome@apps.indivo.org",
  "version" : "2.0.0",
  "indivo_version": "2.0.0",
  "ui_app": true
}
```

5.21.2 App OAuth Credentials

When authenticating to Indivo using traditional OAuth, applications must provided Indivo with their consumer key and a shared consumer_secret. As this secret is private and should not be shared with other apps (i.e., via a call to GET /apps/), it should be registered in a separate file. We therefore define a simple JSON format for specifying app OAuth credentials, which is simple JSON and has two fields:

• **consumer_key**: The OAuth consumer key for the app.
• **consumer_secret**: The oAuth consumer secret for the app.

Here’s a sample credentials file, for our built-in Problems app:

```json
{
    "consumer_key": "problems@apps.indivo.org",
    "consumer_secret": "SECRETFORTHEPROBLEMSAPP:CHANGEME"
}
```

**Notes:**

• If your app is a SMART app, you probably haven’t explicitly generated a ‘consumer key’. You should set the `consumer_key` field of the credentials file to match the `id` field of your app manifest file.

• If your app is a SMART CONNECT app (or connects to Indivo using *In-Browser Connect Authentication*), you do not need a consumer secret. In such a case, set the `consumer_secret` field of the credentials file to the empty string: `''`.

### 5.21.3 Managing the Registered Apps

Thus, to register an app with Indivo, you need two files: an app manifest (`manifest.json`) and a credentials file (`credentials.json`).

Changing the set of registered apps in Indivo is now drag-and-drop, as with our process for managing datatypes and schemas. To add, remove, or change an app, you’ll need to:

• Create a manifest and credentials file for the app (or modify existing manifests/credentials)

• Drop the files into the filesystem

• Sync the database with the filesystem

#### Apps in the Indivo Filesystem

Indivo apps currently have the following layout on the filesystem:

```
indivo_server/
    registered_apps/
        admin/
        ui/
        user/
            allergies/
                manifest.json
                credentials.json
            ...
```

To add an app to the filesystem, simply add a subdirectory under `indivo_server/registered_apps/admin`, `indivo_server/registered_apps/ui`, or `indivo_server/registered_apps/user` (depending on the type of your app), and drop a manifest and a credentials file into that directory.

To remove an app, just delete its directory.

To change an app’s manifest or credentials, just modify the appropriate `manifest.json` or `credentials.json` file.

#### Syncing the Database with the Filesystem

To alert Indivo that you’ve changed the registered apps, run (from `indivo_server/`):
python manage.py sync_apps

This will process the list of registered apps and sync any additions, deletes or updates to the database.

**Resetting Indivo**

With the new system, there is **NO NEED TO RESET INDIVO TO ADD APPS!** Simply run the `sync_apps` command, above.

When you do reset Indivo, the reset script now calls `sync_apps`, which will add all of the registered apps to Indivo.

### 5.22 Sample Data in Indivo

Testing out new Indivo apps is difficult at best with no patient data to run against. That’s why we’ve added the capability to create records with sample patient data pre-loaded into them. There are two pathways for getting sample data into Indivo: using the `indivo_data.xml` file and the `reset.py` script, and putting Indivo into Demo Mode.

#### 5.22.1 Using `indivo_data.xml`

The simplest way to get sample data into Indivo is to use the basic reset script. When you’re setting up `indivo_data.xml`, just add a `data_profile` attribute to any `<record>` tags that you’re creating. The default `indivo_data.xml` file has an example of this already: it loads the data profile for `patient_2` into the John S. Smith record.

As described below, you can determine available data profiles by looking at the subdirectories of `settings.SAMPLE_DATA_DIR`.

#### 5.22.2 Using Demo Mode

If you would like to run Indivo fully populated by sample data (as we do on our developer’s sandbox), you can put Indivo into **Demo Mode**. In this mode, all newly created accounts are immediately set up with records pre-loaded with sample data.

You can do this by configuring the following settings in `settings.py`:

- **SAMPLE_DATA_DIR** The directory where sample data is located.
- **DEMO_MODE** Puts Indivo into Demo Mode if set to True.
- **DEMO_PROFILES** A dictionary mapping record labels to data profiles to load for each new account. For example, if the value of this settings were:

  ```python
  {'John Doe':'patient_1',
   'Robert Frost':'bob',
   'Ted Kennedy':'patient_2',
  }
  ```

Then for each new account, three new records would be created. The first would have a label of ‘John Doe’, and be populated by the data profile ‘patient_1’. The second would have a label of ‘Robert Frost’, and be populated by the data profile ‘bob’. The third would have a label of ‘Ted Kennedy’, and be populated by the data profile ‘patient_2’.

5.22. Sample Data in Indivo
Note: Demo Mode autocreates records any time the API call to create an account, POST /accounts/, is called. This means that any records created through other means (i.e. by a call to POST /records/) will not be populated with data. If your registration UI or admin app handles record creation, this could lead to the existence of some records populated with sample data, and others without it.

5.22.3 Available Sample Data

Any data in the directory specified by settings.SAMPLE_DATA_DIR (settings.APP_HOME/sample_data by default) is available for loading into Indivo. Data in SAMPLE_DATA_DIR should look like:

```
profile_1/   # Data profiles. Each directory should correspond to a single patient.
  bob/       # For example, this is the data profile that can be referenced as 'bob'
    Demographics.xml # An optional demographics document.
  doc_1.xml  # XML Data to load goes here.
  doc_2.xml  # File names MUST be prefixed with 'doc_'.
    ...
  doc_n.xml

doc_1.pdf   # Other extensions are treated as binary docs.
doc_2.pdf   # Also prefix names with 'doc_'.
    ...
```

Namely, the data directory should have multiple subdirectories, each representing one patient's data. Within a patient’s directory, there might be a Demographics.xml file. There will also be any number of data files, labeled doc_{NAME}.{EXTENSION}, where NAME can be anything, and EXTENSION describes the type of data in the file.

We’ve provided you with a few sample patients to get started with, but you should feel free to add data that is useful to your specific Indivo installation.

5.22.4 Adding To the Available Sample Data

Adding sample data to Indivo is trivial: simply add files to settings.SAMPLE_DATA_DIR, making sure to preserve the directory structure described above. You can either:

- Add data to an existing profile, by dropping new data files into that profile’s directory, or
- Add a new profile, by creating a new subdirectory of SAMPLE_DATA_DIR. Make sure to add a demographics document for the new profile.

5.23 Indivo Schemas

5.23.1 Introduction

Schemas in Indivo are used to describe valid formats in which data may enter Indivo. Right now, we use XSDs as schemas, since we accept input data only in XML form, but in the future we might extend this to include schemas for validating other formats of data (OWL for RDF, etc.). Note that schemas describe the format of input data only! For information on how data is processed and stored in Indivo, see Data Pipeline.
There are a number of XML standards for medical activities, ranging from the CCR summary to the highly detailed CCD. None of these are particularly well tuned to the needs of a PCHR, where an individual datum may come from a hospital data feed, or from patient-based data entry. The Indivo schemas are built to serve the specific PCHR needs. Importantly, the Indivo schemas use standard coding systems wherever possible. The schemas are also ready for new coding systems as they emerge, especially in the realm of personally-controlled medicine with simplified terminology.

### 5.23.2 What if I want to store data that doesn’t match an Indivo schema?

Indivo X is designed to accept documents that conform to any XML schema, such as CCR, and even documents that are not XML, i.e. PDFs, MPEG, etc....

XML documents that conform to the built-in schemas can be immediately transformed, via the Indivo X Data Pipeline, into individual datapoints, which can then be queried using the Indivo Reporting API. XML documents that conform to custom schemas are not processed, and therefore cannot be retrieved using the reporting API (though you can still access them with API calls for retrieving unprocessed documents, which will return them in their original XML form.

If you want to extend Indivo to enable querying over data input according to a new schema, see Adding Custom Schemas to Indivo.

### 5.23.3 Namespace and XML Types

All of the default Indivo X document schemas are in a single namespace:

http://indivo.org/vocab/xml/documents#

The use of the trailing # enables simple RDF-like concatenation of namespace prefix and suffix to generate a single type URL. For example, an SDMX document in the Indivo documents namespace will have as its type:

http://indivo.org/vocab/xml/documents#Models

### 5.23.4 Design Rationale for Inclusion vs. Relation

Indivo X brings the ability to relate documents to one another using metadata, rather than document payload. This is particularly important when the payload might not be under the user’s control, i.e. a CCR document. It can also be useful even in the design of new Indivo schemas.

One could imagine separation the prescription information from the medication information, having two documents related to one another rather than one bigger document. However, our design rationale for now is to keep medication and its prescription data in the same XML document because those two chunks of data are generated in the same event. If, at some point, Indivo stores prescription filling information, then it is likely that this information would be more appropriately stored in a separate, linked document.

### 5.23.5 Core Schemas

All schema files and sample instance documents are available at http://indivo.org/vocab/xml/. Note that these schemas are only the ones that come with Indivo by default. Each instance of Indivo might define additional, custom schemas that are not documented here. See Adding Custom Schemas to Indivo for instructions on how to add custom schemas to Indivo.
Metadata and Indivo Internal Data Structures

Indivo Document Metadata Schema

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- didn’t place this in the Indivo namespace because it’s not medical payload -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:complexType name="Principal">
   <xs:sequence>
      <xs:element name="fullname" type="xs:string" minOccurs="1" maxOccurs="1" />
   </xs:sequence>
   <!-- e.g. "fhh@apps.indivo.org" -->
   <xs:attribute name="id" type="xs:string" use="required" />
   <!-- e.g. "userapp" or "account" or "adminapp" -->
   <xs:attribute name="type" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType name="Relation">
   <xs:attribute name="type" type="xs:string" use="required" />
   <xs:attribute name="count" type="xs:integer" use="required" />
</xs:complexType>

<xs:element name="Document">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="createdAt" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
         <xs:element name="creator" type="Principal" minOccurs="1" maxOccurs="1" />
         <!-- if suppressedAt is non-null, then suppressor should be present -->
         <xs:element name="suppressedAt" type="xs:dateTime" minOccurs="0" maxOccurs="1" />
         <xs:element name="suppressor" type="Principal" minOccurs="0" maxOccurs="1" />
         <xs:element name="replacedBy" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:attribute name="id" type="xs:string" use="required" />
            </xs:complexType>
         </xs:element>
         <xs:element name="replaces" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:attribute name="id" type="xs:string" use="required" />
            </xs:complexType>
         </xs:element>
         <xs:element name="original" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:attribute name="id" type="xs:string" use="required" />
            </xs:complexType>
         </xs:element>
         <xs:element name="latest" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:attribute name="id" type="xs:string" use="required" />
               <xs:attribute name="createdAt" type="xs:dateTime" use="required" />
               <xs:attribute name="createdBy" type="xs:string" use="required" />
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```
Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Document id="14c81023-c84f-496d-8b8e-9438280441d3" type="" digest="7e9bc09276e0829374fd810f96ed98d5" size="77"
<createdAt>2009-05-04T17:05:33</createdAt>
<creator id="steve@indivo.org" type="account">
  <fullname>Steve Zabak</fullname>
</creator>
<suppressedAt>2009-05-06T17:05:33</suppressedAt>
<suppressor id="steve@indivo.org" type="account">
  <fullname>Steve Zabak</fullname>
</suppressor>
<original id="14c81023-c84f-496d-8b8e-9438280441d3" />
<latest id="14c81023-c84f-496d-8b8e-9438280441d3" createdAt="2009-05-05T17:05:33" createdBy="steve@indivo.org">
  <label>HBA1C reading</label>
  <status>active</status>
  <nevershare>false</nevershare>
  <relatesTo>
    <relation type="http://indivo.org/vocab/documentrels#attachment" count="1" />
    <relation type="http://indivo.org/vocab/documentrels#annotation" count="5" />
  </relatesTo>
  <isRelatedFrom>
    <relation type="http://indivo.org/vocab/documentrels#interpretation" count="1" />
  </isRelatedFrom>
</latest>
</Document>
```
Indivo Account Schema

An Indivo Account represents a single user of the system, with their basic info and the ways in which they authenticate. It is separate from a record.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <element name="Account">
    <complexType>
      <sequence>
        <element name="secret" type="string" minOccurs="0" maxOccurs="1" />
        <element name="fullName" type="string" minOccurs="1" maxOccurs="1" />
        <element name="contactEmail" type="string" minOccurs="1" maxOccurs="1" />
        <element name="lastLoginAt" type="dateTime" minOccurs="1" maxOccurs="1" />
        <element name="totalLoginCount" type="integer" minOccurs="1" maxOccurs="1" />
        <element name="failedLoginCount" type="integer" minOccurs="1" maxOccurs="1" />
        <element name="state" type="string" minOccurs="1" maxOccurs="1" />
        <element name="lastStateChange" type="dateTime" minOccurs="1" maxOccurs="1" />
        <element name="authSystem" minOccurs="0" maxOccurs="unbounded">
          <complexType>
            <attribute name="name" type="string" use="required" />
            <attribute name="username" type="string" use="required" />
          </complexType>
        </element>
      </sequence>
      <attribute name="id" type="string" use="required" />
    </complexType>
  </element>
</schema>
```

Example:

```xml
<Account id="joeuser@indivo.example.org">
  <fullName>Joe User</fullName>
  <contactEmail>joeuser@gmail.com</contactEmail>
  <lastLoginAt>2010-05-04T15:34:23Z</lastLoginAt>
  <totalLoginCount>43</totalLoginCount>
  <failedLoginCount>0</failedLoginCount>
  <state>active</state>
  <lastStateChange>2009-04-03T12:12Z</lastStateChange>
  <authSystem name="password" username="joeuser" />
  <authSystem name="hospital_sso" username="Joe_User" />
</Account>
```

Indivo PHA Schema

Information describing a Personal Health App (User App). Can be wrapped into a set of Apps.

Schema:
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:complexType name="App">
    <xs:sequence>
      <xs:element name="startURLTemplate" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="description" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="autonomous" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      <xs:element name="autonomousReason" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="frameable" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
      <xs:element name="ui" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required"/>
  </xs:complexType>

  <xs:element name="App" type="App"/>

  <xs:element name="Apps">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="App" type="App"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

Example:

<App id="problems@apps.indivo.org">
  <startURLTemplate>http://problems.indivo.org/auth/start?record_id={record_id}&amp;carenet_id={carenet_id}</startURLTemplate>
  <name>Problem List</name>
  <description>Managing your problem list</description>
  <autonomous>false</autonomous>
  <frameable>true</frameable>
  <ui>true</ui>
</App>

Example of multiple apps:

<Apps>
  <App id="problems@apps.indivo.org">
    <startURLTemplate>http://problems.indivo.org/auth/start?record_id={record_id}&amp;carenet_id={carenet_id}</startURLTemplate>
    <name>Problem List</name>
    <description>Managing your problem list</description>
    <autonomous>false</autonomous>
    <frameable>true</frameable>
    <ui>true</ui>
  </App>
</Apps>

**Indivo Audit Log Schema**

As of Beta 3, the logs will be returned as Indivo Reports according to the *Indivo Reporting Schema*. Each report item will be of type `<AuditEntry>`, as defined below:
Example:

<?xml version="1.0" encoding="utf-8" ?>
<AuditEntry>
  <BasicInfo datetime="2011-04-27T17:32:23Z" view_func="get_document" request_successful="true" />
  <PrincipalInfo effective_principal="myapp@apps.indivohealth.org" proxied_principal="me@indivohealth.org" />
  <Resources carenet_id="" record_id="123" pha_id="" document_id="acd" external_id="" message_id="" />
  <RequestInfo req_url="/records/123/documents/acd/" req_ip_address="127.0.0.1" req_domain="localhost" req_method="GET" />
  <ResponseInfo resp_code="200" />
</AuditEntry>
Indivo Carenet Schema

A list of carenets is returned when a user/app wants to know how a document is shared. However, this same list of carenets might be used in a different setting. Thus, the “mode” attribute is optional. It indicates whether sharing in this carenet was done explicitly, or via some implicit auto-share rule.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="Carenets">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Carenet" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required" />
            <xs:attribute name="name" type="xs:string" use="required" />
            <xs:attribute name="mode" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="record_id" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```xml
<Carenets record_id="123">
  <Carenet id="789" name="Work/School" mode="explicit" />
</Carenets>
```

Indivo Document Status History Schema

When a document’s status changes (archived, etc.), its history of changes is documented and available in this schema.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="DocumentStatusHistory">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DocumentStatus" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="reason" type="xs:string" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
            <xs:attribute name="by" type="xs:string" use="required" />
            <xs:attribute name="at" type="xs:dateTime" use="required" />
            <xs:attribute name="status" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="document_id" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Indivo X Documentation, Release 2.0.0

Example:

<DocumentStatusHistory document_id="456">
  <DocumentStatus by="joeuser@indivo.example.org" at="2010-09-03T12:45:12Z" status="archived">
    <reason>no longer relevant</reason>
  </DocumentStatus>
</DocumentStatusHistory>

Indivo Inbox Message Schema

Indivo messages, sent to accounts (sometimes via a record), are represented with the following schema.

Schema:

<x:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:complexType name="Message">
    <xs:sequence>
      <xs:element name="sender" type="xs:string" minOccurs="1" maxOccurs="1" />  
      <xs:element name="received_at" type="xs:dateTime" minOccurs="1" maxOccurs="1" />  
      <xs:element name="read_at" type="xs:dateTime" minOccurs="0" maxOccurs="1" />  
      <xs:element name="subject" type="xs:string" minOccurs="1" maxOccurs="1" />  
      <xs:element name="record" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required" />  
        </xs:complexType>
      </xs:element>
      <xs:element name="attachment" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="num" type="xs:integer" use="required" />  
          <xs:attribute name="type" type="xs:string" use="required" />  
          <xs:attribute name="size" type="xs:integer" use="required" />  
          <xs:attribute name="doc_id" type="xs:string" />  
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />  
  </xs:complexType>
  <xs:element name="Message" type="Message" />
</xs:complexType>

Example:

<x:element name="Message" type="Message" />
<x:element name="Messages">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Message" type="Message" minOccurs="0" maxOccurs="unbounded" />  
    </xs:sequence>
  </xs:complexType>
</xs:element>
<Messages>
  <Message id="879">
    <sender>doctor@example.indivo.org</sender>
    <received_at>2010-09-04T14:12:12Z</received_at>
    <read_at>2010-09-04T17:13:24Z</read_at>
    <subject>your test results are looking good</subject>
    <severity>normal</severity>
    <record id="123" />
    <attachment num="1" type="http://indivo.org/vocab/xml/documents#Lab" size="12546" />
  </Message>
</Messages>

Another Example:

<Messages>
  <Message id="879">
    <sender>doctor@example.indivo.org</sender>
    <received_at>2010-09-04T14:12:12Z</received_at>
    <subject>your test results are looking good</subject>
    <severity>normal</severity>
    <record id="123" />
    <attachment num="1" type="http://indivo.org/vocab/xml/documents#Lab" size="12546" />
  </Message>
</Messages>

**Indivo Notification Schema**

The Indivo Healthfeed includes notifications, represented by this schema:

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="Notifications">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Notification">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="sender" type="xs:string" minOccurs="1" maxOccurs="1" />
              <xs:element name="received_at" type="xs:dateTime" minOccurs="1" maxOccurs="1" />
              <xs:element name="content" type="xs:string" minOccurs="1" maxOccurs="1" />
              <xs:element name="record" minOccurs="1" maxOccurs="1">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:string" use="required" />
                  <xs:attribute name="label" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
              <xs:element name="document" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:string" use="required" />
                  <xs:attribute name="label" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5.23. Indivo Schemas
Example:

```xml
<Notifications>
  <Notification id="468">
    <sender> labs@apps.indivo.org </sender>
    <received_at> 2010-09-03T15:12:12Z </received_at>
    <content> A new lab result has been delivered to your account </content>
    <record id="123" label="Joe User" />
    <document id="579" label="Lab Test 2" />
  </Notification>
</Notifications>
```

**Indivo Permissions Schema**

Coming Soon...

**Indivo Record Schema**

The basic info for an Indivo record. Some of the attributes are there for indicating sharing relationships.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="Record">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="demographics" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="document_id" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="created" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="at" type="xs:dateTime" use="optional" />
            <xs:attribute name="by" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required" />
      <xs:attribute name="label" type="xs:string" use="required" />
      <xs:attribute name="shared" type="xs:boolean" use="optional" />
      <xs:attribute name="carenet_id" type="xs:string" use="optional" />
      <xs:attribute name="carenet_name" type="xs:string" use="optional" />
      <xs:attribute name="role_label" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Indivo Request Token Schema

The Indivo UI Server needs to manage request tokens for apps so that it can display the appropriate authorization screens. This schema makes use of the Indivo PHA Schema.

Schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:include schemaLocation="../pha/pha.xsd" />

  <xs:element name="RequestToken">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="record" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
        <xs:element name="carenet" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
        <xs:element name="kind" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="App" type="App" minOccurs="1" maxOccurs="1" />
      </xs:sequence>
      <xs:attribute name="token" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example:

```xml
<RequestToken token="XYZ">
  <record id="123" />
  <carenet />
  <kind>new</kind>
  <App id="problems@apps.indivo.org">
    <name>Problem List</name>
    <description>Managing your list of problems</description>
    <autonomous>false</autonomous>
</App>
</RequestToken>
```
Indivo Coded Values

A coded value is a value taken from a coding system. It consists of a reference to the coding system (a URL), the code value, and the human-readable string. When the coding system is not used but a manual value is entered, the coding system and coded value are absent, leaving only the human-readable string.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
  <xs:complexType name="CodedValue">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="type" type="xs:anyURI" use="optional" />
        <xs:attribute name="value" type="xs:string" use="optional" />
        <xs:attribute name="abbrev" type="xs:string" use="optional" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

- When a document comes into Indivo, its coded values may be expanded (with abbreviation and element content) or not (just the code and coding system).
- We will encourage applications to provide expanded coded values, but this will not be required.
- Reports will provide abbreviations and full names for all relevant codes by looking up against Indivo-stored copies of the coding systems. Documents will not be modified from what the sources send us, to follow the principles of store exactly the original data source (that’s required because the documents might be digitally signed.)
- Reports can flag codes whose abbreviations and full names do not match the coding system data (but we always show by default what the document says, we trust the source, not the coding system.)
- We then need a code lookup API for viewing single documents.
- codes.indivo.org will provide an API for interpreting codes.

Indivo Data Values Schema

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
  <xs:complexType name="Range">
    <xs:sequence>
      <!-- a missing minimum means < max -->
      <xs:element name="minimum" type="xs:double" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
</xs:schema>
```
<!-- a missing maximum means > min -->
<xs:element name="maximum" type="xs:double" minOccurs="0" maxOccurs="1" />

<!-- technically this schema allows a range with neither min nor max, which doesn’t mean much, -->
<xs:element name="unit" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
</xs:sequence>
</xs:complexType>

<!-- an ordinal, i.e. "2+" is coded using textValue -->
<xs:complexType name="ValueAndUnit">
  <xs:sequence>
    <xs:element name="value" type="xs:double" minOccurs="0" maxOccurs="1" />
    <xs:element name="textValue" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="unit" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<!-- a result is an abstract base, only subtypes can be used -->
<xs:complexType name="Result" abstract="true">
  <xs:sequence>
    <!-- HL7 defines flag types -->
    <xs:element name="flag" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ResultInRange">
  <xs:complexContent>
    <xs:extension base="indivo:Result">
      <xs:sequence>
        <xs:element name="valueAndUnit" type="indivo:ValueAndUnit" minOccurs="1" maxOccurs="1" />
        <xs:element name="normalRange" type="indivo:Range" minOccurs="0" maxOccurs="1" />
        <!-- nontoxicrange as in, if it’s outside the range, it’s toxic -->
        <xs:element name="nonCriticalRange" type="indivo:Range" minOccurs="0" maxOccurs="1" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- untyped strings, e.g. "positive" -->
<xs:complexType name="ResultInSet">
  <xs:complexContent>
    <xs:extension base="indivo:Result">
      <xs:sequence>
        <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="option" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="normal" type="xs:boolean" use="required" />
                <xs:attribute name="description" type="xs:string" use="optional" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
Indivo X Documentation, Release 2.0.0

<xs:complexType>
  <xs:element>
  </xs:element>
</xs:complexType>

<xs:complexType name="ValueOrRange">
  <xs:choice>
    <xs:element name="value" type="indivo:ValueAndUnit" />
    <xs:element name="range" type="indivo:Range" />
  </xs:choice>
</xs:complexType>

<xs:complexType name="Concentration">
  <xs:complexContent>
    <xs:extension base="indivo:ValueOrRange" />
  </xs:complexContent>
</xs:complexType>

Indivo Provider Schema

A provider is typically an MD with an institution affiliation.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
  xmlns:indivo="http://indivo.org/vocab/xml/documents#"
  elementFormDefault="qualified">
<complexType name="Provider">
  <sequence>
    <element name="name" type="string" minOccurs="1" maxOccurs="1" />
    <element name="institution" type="string" minOccurs="1" maxOccurs="1" />
  </sequence>
</complexType>

<complexType name="LabProvider">
  <sequence>
    <element name="name" type="string" minOccurs="1" maxOccurs="1" />
    <element name="address" type="string" minOccurs="1" maxOccurs="1" />
  </sequence>
</complexType>

<complexType name="Signature">
  <sequence>
    <element name="at" type="dateTime" minOccurs="0" maxOccurs="1" />
    <element name="provider" type="indivo:Provider" minOccurs="1" maxOccurs="1" />
  </sequence>
</complexType>

</schema>
```
Indivo X Documentation, Release 2.0.0

Reporting

Indivo Reporting Schema

See the schema for Indivo Document Metadata Schema and specific indivo:doc schemas.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
    elementFormDefault="qualified">

    <xs:complexType name="Report">
        <xs:sequence>
            <xs:element name="Meta" minOccurs="1" maxOccurs="1">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Document" minOccurs="1" maxOccurs="1" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Item" minOccurs="1" maxOccurs="1">
                <xs:complexType>
                    <xs:sequence>
                        <xs:choice minOccurs="1" maxOccurs="1">
                            <xs:element name="Allergy" />
                            <xs:element name="Equipment" />
                            <xs:element name="Immunization" />
                            <xs:element name="LabReport" />
                            <xs:element name="Medication" />
                            <xs:element name="Problem" />
                            <xs:element name="Procedure" />
                            <xs:element name="SimpleClinicalNote" />
                            <xs:element name="VitalSign" />
                            <xs:element name="AggregateReport" />
                        </xs:choice>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>

    <xs:element name="Reports">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Summary" minOccurs="1" maxOccurs="1">
                    <xs:complexType>
                        <xs:attribute name="total_document_count" use="optional" type="xs:int" />
                        <xs:attribute name="limit" use="optional" type="xs:int" />
                        <xs:attribute name="offset" use="optional" type="xs:int" />
                        <xs:attribute name="order_by" use="optional" type="xs:string" />
                    </xs:complexType>
                </xs:element>
                <xs:element name="QueryParams" minOccurs="1" maxOccurs="1">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="GroupBy" minOccurs="0" maxOccurs="1">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="GroupBy" minOccurs="0" maxOccurs="1">
                                            <xs:complexType>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

5.23. Indivo Schemas 597
Example:

<?xml version="1.0" encoding="utf-8" ?>
<Reports xmlns="http://indivo.org/vocab/xml/documents#">
  <Summary total_document_count="2" limit="100" offset="0" order_by="date_measured" />
  <QueryParams>
    <DateRange value="date_measured*1995-03-10T00:00:00Z*" />
    <Filters>
      <Filter name="lab_type" value="hematology"/>
    </Filters>
  </QueryParams>
  <Report>
    <Meta>
      <Document id="261ca370-927f-41af-b001-7b615c7a468e" type="http://indivo.org/vocab/xml/documents#Lab" size="1653" digest="0799971784e5a2d199cd6585415a8cd57f7bf9e4f8c8f74ef67a1009a1481cd6" record_id=""></Document>
    </Meta>
  </Report>
</Reports>
<label>testing</label>
<status>active</status>
<nevershare>false</nevershare>
</Document>
</Meta>
<Item>
<LabReport xmlns="http://indivo.org/vocab/xml/documents#">
<dateMeasured>1998-07-16T12:00:00Z</dateMeasured>
<labType>hematology</labType>
<laboratory>
  <name>Quest</name>
  <address>300 Longwood Ave, Boston MA 02215</address>
</laboratory>
<comments>was looking pretty sick</comments>
<firstPanelName>CBC</firstPanelName>
</LabReport>
</Item>
</Report>
<Report>
<Meta>
<Document id="1b7270a6-5925-450c-9273-5a74386cef63" type="http://indivo.org/vocab/xml/documents#Lab" size="1653" digest="c1be22813ab83f6b3858878a802f372eef754fcdd285e44a5fdb7387d6ee3667" record_id="">
<createdAt>2011-05-02T17:48:13Z</createdAt>
<creator id="mymail@mail.ma" type="Account">
  <fullname>full name</fullname>
</creator>
<original id="1b7270a6-5925-450c-9273-5a74386cef63"/>
<label>testing</label>
<status>active</status>
<nevershare>false</nevershare>
</Document>
</Meta>
<Item>
<LabReport xmlns="http://indivo.org/vocab/xml/documents#">
<dateMeasured>2009-07-16T12:00:00Z</dateMeasured>
<labType>hematology</labType>
<laboratory>
  <name>Quest</name>
  <address>300 Longwood Ave, Boston MA 02215</address>
</laboratory>
<comments>was looking pretty sick</comments>
<firstPanelName>CBC</firstPanelName>
</LabReport>
</Item>
</Report>
</Reports>

**Indivo Aggregate Report Schema**

This schema describes report items returned in aggregate form.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
  elementFormDefault="qualified">
```
Indivo X Documentation, Release 2.0.0

<xs:complexType name="AggregateReport">
  <xs:attribute name="value" type="xs:string" use="required" />
  <xs:attribute name="group" type="xs:string" use="optional" />
</xs:complexType>

<xs:element name="AggregateReport" type="indivo:AggregateReport" />
</xs:schema>

Example:

<?xml version="1.0" encoding="utf-8" ?>

Indivo Generic Aggregate Reports Schema

This schema describes report items returned in XML aggregate form from Generic Reports.

Schema:

<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="AggregateReport">
    <xs:attribute name="value" type="xs:string" use="required" />
    <xs:attribute name="group" type="xs:string" use="optional" />
  </xs:complexType>

  <xs:element name="AggregateReports">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="AggregateReport" type="AggregateReport" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

Example:

<?xml version="1.0" encoding="utf-8" ?>
<AggregateReports>
  <AggregateReport value="1" group="2009-07" />
  <AggregateReport value="4" group="2009-08" />
</AggregateReports>

Special Documents

Indivo Document Demographics Schema

Schema:

<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:indivo="http://indivo.org/vocab/xml/documents#"
  elementFormDefault="qualified"
  targetNamespace="http://indivo.org/vocab/xml/documents#">
<xs:complexType name="Name">
  <xs:sequence>
    <xs:element name="familyName" type="xs:string" minOccurs="1" maxOccurs="1" />
    <xs:element name="givenName" type="xs:string" minOccurs="1" maxOccurs="1" />
    <xs:element name="middleName" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="prefix" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="suffix" type="xs:string" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="PhoneType">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="h"/>
    <xs:enumeration value="w"/>
    <xs:enumeration value="c"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="GenderType">
  <xs:restriction base="xs:normalizedString">
    <xs:enumeration value="female"/>
    <xs:enumeration value="male"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="Telephone"> 
  <xs:sequence>
    <xs:element name="type" type="indivo:PhoneType" minOccurs="0" maxOccurs="1" />
    <xs:element name="number" type="xs:string" minOccurs="1" maxOccurs="1" />
    <xs:element name="preferred" type="xs:boolean" minOccurs="0" maxOccurs="1" default="false" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Address">
  <xs:sequence>
    <xs:element name="country" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="city" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="postalCode" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="region" type="xs:string" minOccurs="0" maxOccurs="1" />
    <xs:element name="street" type="xs:string" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>

<xs:element name="Demographics">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="dateOfBirth" type="xs:date" minOccurs="1" />
      <xs:element name="gender" type="indivo:GenderType" minOccurs="1" />
      <xs:element name="email" type="xs:string" minOccurs="0" />
      <xs:element name="ethnicity" type="xs:string" minOccurs="0" />
      <xs:element name="preferredLanguage" type="xs:string" minOccurs="0" />
      <xs:element name="race" type="xs:string" minOccurs="0" />
      <xs:element name="Name" type="indivo:Name" minOccurs="1" />
      <xs:element name="Telephone" type="indivo:Telephone" minOccurs="0" maxOccurs="2" />
      <xs:element name="Address" type="indivo:Address" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
Example:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Demographics xmlns="http://indivo.org/vocab/xml/documents#">
    <dateOfBirth>1939-11-15</dateOfBirth>
    <gender>male</gender>
    <email>test@fake.org</email>
    <ethnicity>Scottish</ethnicity>
    <preferredLanguage>EN</preferredLanguage>
    <race>caucasian</race>
    <Name>
        <familyName>Wayne</familyName>
        <givenName>Bruce</givenName>
        <middleName>Quentin</middleName>
        <prefix>Mr</prefix>
        <suffix>Jr</suffix>
    </Name>
    <Telephone>
        <type>h</type>
        <number>555-5555</number>
        <preferred>true</preferred>
    </Telephone>
    <Telephone>
        <type>c</type>
        <number>555-6666</number>
    </Telephone>
    <Address>
        <country>USA</country>
        <city>Gotham</city>
        <postalCode>90210</postalCode>
        <region>secret</region>
        <street>1007 Mountain Drive</street>
    </Address>
</Demographics>
```

**Indivo Document Annotation Schema**

Medical documents can be annotated in Indivo, with a document that’s added in relation to the annotated document. This is a simple schema for these text-based annotations.

Schema:

Coming Soon!

Example:

Coming Soon!

The relationship to the annotated document is maintained in Indivo metadata, and the annotation can, optionally, store the SHA256 hash of the referenced document for robustness.
Medical Documents

Indivo Document Schema: Procedure

A procedure is effectively a surgical event (though some are not exactly surgical.)

See also the schema for Indivo Coded Values and for Indivo Provider Schema.

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://indivo.org/vocab/xml/documents#"
    xmlns:indivo="http://indivo.org/vocab/xml/documents#"
    elementFormDefault="qualified">
    <include schemaLocation="../../common/codes.xsd" />
    <include schemaLocation="../../common/provider.xsd" />
    <element name="Procedure">
      <complexType>
        <sequence>
          <element name="datePerformed" type="dateTime" minOccurs="0" maxOccurs="1" />
          <element name="name" type="indivo:CodedValue" minOccurs="1" maxOccurs="1" />
          <element name="provider" type="indivo:Provider" minOccurs="0" />
          <element name="location" type="string" minOccurs="0" maxOccurs="1" />
          <element name="comments" type="string" minOccurs="0" maxOccurs="1" />
        </sequence>
      </complexType>
    </element>
</schema>
```

Example:

```xml
<Procedure xmlns="http://indivo.org/vocab/xml/documents#">
  <datePerformed>2009-05-16T12:00:00</datePerformed>
  <name type="http://codes.indivo.org/procedures#" value="85" abbrev="append">Appendectomy</name>
  <provider>
    <name>Kenneth Mandl</name>
    <institution>Children’s Hospital Boston</institution>
  </provider>
</Procedure>
```

Indivo Document Schema: Equipment

Schema:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://indivo.org/vocab/xml/documents#"
    xmlns:indivo="http://indivo.org/vocab/xml/documents#"
    elementFormDefault="qualified">
    <element name="Equipment">
      <complexType>
        <sequence>
          <element name="dateStarted" type="date" minOccurs="0" maxOccurs="1" />
          <element name="dateStopped" type="date" minOccurs="0" maxOccurs="1" />
          <element name="type" type="string" minOccurs="0" maxOccurs="1" />
          <element name="name" type="string" minOccurs="1" maxOccurs="1" />
        </sequence>
      </complexType>
    </element>
</schema>
```
Example:

```
<Equipment xmlns="http://indivo.org/vocab/xml/documents#">
  <dateStarted>2009-02-05</dateStarted>
  <dateStopped>2010-06-12</dateStopped>
  <type>cardiac</type>
  <name>Pacemaker</name>
  <vendor>Acme Medical Devices</vendor>
  <id>167-ABC-23</id>
  <description>it works</description>
  <specification>blah blah blah</specification>
</Equipment>
```

Indivo Document Schema: Simple Clinical Note

A full clinical note needs to contain a number of coded problems, etc. Some hospital systems do not have fully normalized clinical notes, in which case they can use this schema to store some simple attributes and the main free-form text of the note.

See also Indivo Coded Values and Indivo Provider Schema.

Schema:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
  elementFormDefault="qualified">
  <include schemaLocation="../../common/codes.xsd" />
  <include schemaLocation="../../common/provider.xsd" />

  <!-- this is mostly a chunk o' text. More normalized clinical notes will be in a diff schema -->
  <element name="SimpleClinicalNote">
    <complexType>
      <sequence>
        <element name="dateOfVisit" type="dateTime" minOccurs="1" maxOccurs="1" />
        <element name="finalizedAt" type="dateTime" minOccurs="0" maxOccurs="1" />
        <element name="visitType" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
        <element name="visitLocation" type="string" minOccurs="0" maxOccurs="1" />
        <element name="specialty" type="indivo:CodedValue" minOccurs="0" maxOccurs="1" />
        <element name="signature" type="indivo:Signature" minOccurs="1" maxOccurs="unbounded" />
        <element name="chiefComplaint" type="string" minOccurs="0" maxOccurs="1" maxOccurs="unbounded" />
        <element name="content" minOccurs="0" maxOccurs="1" type="string" />
      </sequence>
    </complexType>
  </element>
</schema>
```

Example:
Indivo X Documentation, Release 2.0.0

Indivo Document Schema: SDMX

For any data model in Indivo that can be represented in SDML, we will accept data in the form of SDMX.

Schema:

```xml
<complexType name="ModelType">
  <sequence>
    <element name="Field" minOccurs="0" maxOccurs="unbounded">
      <complexType mixed="true">
        <choice>
          <element name="Model" type="indivo:ModelType" minOccurs="0" maxOccurs="1"/>
          <element name="Models" type="indivo:ModelsType" minOccurs="0" maxOccurs="1"/>
        </choice>
        <attribute name="name" type="string" use="required"/>
      </complexType>
    </element>
  </sequence>
  <attribute name="name" type="string" use="required" />
  <attribute name="documentId" type="string" use="optional" />
</complexType>

<complexType name="ModelsType">
  <sequence>
    <element name="Model" type="indivo:ModelType" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```
Example:

```xml
<Models xmlns="http://indivo.org/vocab/xml/documents#">
  <Model name="TestMedication" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
    <Field name="date_started">2010-10-01T00:00:00Z</Field>
    <Field name="name">ibuprofen</Field>
    <Field name="brand_name">Advil</Field>
    <Field name="date_stopped">2010-10-31T00:00:00Z</Field>
    <Field name="prescription">
      <Model name="TestPrescription" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
        <Field name="prescribed_by_name">Kenneth D. Mandl</Field>
        <Field name="prescribed_by_institution">Children’s Hospital Boston</Field>
        <Field name="prescribed_on">2010-09-30T00:00:00Z</Field>
        <Field name="prescribed_stop_on">2010-10-31T00:00:00Z</Field>
      </Model>
    </Field>
    <Field name="fills">
      <Models>
        <Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
          <Field name="date_filled">2010-10-01T00:00:00Z</Field>
          <Field name="supply_days">15</Field>
          <Field name="filled_at_name">CVS</Field>
        </Model>
        <Model name="TestFill" documentId="b1d83191-6edd-4aad-be4e-63117cd4c660">
          <Field name="date_filled">2010-10-16T00:00:00Z</Field>
          <Field name="supply_days">15</Field>
          <Field name="filled_at_name">CVS</Field>
        </Model>
      </Models>
    </Field>
  </Model>
</Models>
```

5.23.6 Adding Custom Schemas to Indivo

As of version 1.1 of Indivo X, we’ve added a feature that makes it much easier to add (in a drag-and-drop fashion) new supported schemas to an instance of Indivo. Adding a new schema to Indivo involves:

- Creating the schema
- Mapping the schema to Indivo’s Data Models
- Dropping the schema into the filesystem

Creating the Schema

Indivo currently accepts schemas only in XSD form. There are numerous tutorials and tools on the web to help you create an XSD, so we won’t presume to tell you how you should do it. What matters is that you build an XSD which can validate documents for further processing in the Indivo data pipeline.
Mapping the Schema to Data Models

In order to arrive at processed facts that can be queried and retrieved, you’ll need to have a way to transform documents matching your schema into a form understood by Indivo. We call this tool (intuitively) a Transform, and you can learn how to build one [here](#).

If the data in your new schema doesn’t fit into any of the Indivo Data Models, and you want to add a new data-model to Indivo, see Adding Custom Data-Models to Indivo.

Dropping the Schema into the Filesystem

Indivo schemas currently have the following layout on the filesystem:

```
indivo_server/
   indivo/
       ...
       schemas/
       util/
       metadata/
       data/
       common/
       output/
       core/
       sdmx/
       schema.xsd
       transform.[xslt | py]
       sdmx.xml
       ...
       contrib/
```

The `indivo/schemas/data/core/` directory contains all of our built-in schemas, and you shouldn’t modify it. Since you are ‘contributing’ a schema to Indivo, add your schema to the `indivo/schemas/data/contrib/` directory. Simply:

- Create a new subdirectory under `indivo/schemas/data/contrib/`.
- Drop the following files into that directory:
  - `schema.xsd`: Your schema. This file **MUST BE NAMED SCHEMA.XSD** to be identified as a schema.
  - `transform.xslt` or `transform.py`: Your transform. This file **MUST BE NAMED ‘transform’** to be identified.
  - `sample.xml` *(optional)*: A sample document that should validate against your schema. This is optional, but is a good way to make sure your schema works as intended. If you have one or more sample xml files in your directory (you can name them anything, as long as the filename ends in ‘.xml’), you can make sure that they all validate by running:
    ```
    cd indivo_server/indivo/schemas
    python utils/validate.py
    ```
    This will validate all sample documents against their schema: you should see ‘ok’ at the end of each line of output if there were no errors.
- Restart Indivo for your changes to take effect. Your final directory structure should now look something like:

```
indivo_server/
   indivo/
       ...
       schemas/
```
5.24 Indivo Transforms

5.24.1 Introduction

For the pipeline to be functional, data must be transformed from its original format into processed medical facts ready to be stored in the database. Each schema in Indivo therefore defines a transform that can be applied to any document that validates against the schema.

5.24.2 Transform Outputs

The ultimate output of the transformation step in the data pipeline is a set of Fact objects ready for storage in the database. However, technologies like XSLT are incapable of producing python objects as output. We looked around for a simple, standard way of modeling data that would meet our needs, and came up empty (though we’re open to suggestions if you think you have the silver bullet). As a result, we’ve created our own language, Indivo Simple Data Modeling Language (SDML), to both define our data models and represent documents (in XML or JSON) that match them.

Thus, transforms may output data in any of the following formats:

- Simple Data Model JSON (SDMJ)
- Simple Data Model XML (SDMX)
- Python Fact objects

Outputs are validated on a per-datamodel-basis. For data model definitions and example outputs, see Indivo Data Models.

5.24.3 Types of Transforms

Indivo currently accepts Transforms in two formats:

- XSLT documents
- Python classes
This may change as Indivo begins to accept data in more, varied formats.

XSLTs

We won’t cover XSLTs in any detail here, as their format and use is clearly outlined in the specification. Since XSLT is traditionally used to transform XML to XML, the most natural output format for XSLTs is SDMX.

Python

For those unskilled in the arts of XSLT, we also allow transforms to be defined using python. To define a transform, simply subclass `indivo.document_processing.BaseTransform` and define a valid transformation method. Valid methods are:

```python
BaseTransform.to_facts(doc_etree)
```

Transform an etree into a list of Indivo Fact objects.

Subclasses should implement this method, which takes an `lxml.etree._ElementTree` (the result of calling `etree.parse()`), and returns a list of `indivo.models.Fact` subclasses.

As an example, here’s how you might implement this for a transform from our Simple Clinical Note Schema to our Simple Clinical Note Data Model:

```python
from indivo.document_processing import BaseTransform
from indivo.models import SimpleClinicalNote
from lxml import etree

NS = "http://indivo.org/vocab/xml/documents#"

class Transform(BaseTransform):
    def to_facts(self, doc_etree):
        args = self._get_data(doc_etree)

        # Create the fact and return it
        # Note: This method must return a list
        return [SimpleClinicalNote(**args)]

    def _tag(self, tagname):
        return "{%s}%s"%(NS, tagname)

    def _get_data(self, doc_etree):
        """ Parse the etree and return a dict of key-value pairs for object construction. """
        ret = {}
        _t = self._tag

        # Get the date_of_visit
        ret['date_of_visit'] = doc_etree.findtext(_t('dateOfVisit'))

        # Get the date finalized
        ret['finalized_at'] = doc_etree.findtext(_t('finalizedAt'))

        # Get the visit_type
        visit_type_node = doc_etree.find(_t('visitType'))
        ret['visit_type'] = visit_type_node.text
        ret['visit_type_type'] = visit_type_node.get('type')
        ret['visit_type_value'] = visit_type_node.get('value')
        ret['visit_type_abbrev'] = visit_type_node.get('abbrev')
```

5.24. Indivo Transforms 609
# Get the visit location
ret["visit_location"] = doc_etree.findtext(_t('visitLocation'))

# Get the specialty of the clinician
specialty_node = doc_etree.find(_t('specialty'))
ret["specialty"] = specialty_node.text
ret["specialty_type"] = specialty_node.get('type')
ret["specialty_value"] = specialty_node.get('value')
ret["specialty_abbrev"] = specialty_node.get('abbrev')

# get the signature
signature_node = doc_etree.find(_t('signature'))
provider_node = signature_node.find(_t('provider'))
ret["signed_at"] = signature_node.findtext(_t('at'))
ret["provider_name"] = provider_node.findtext(_t('name'))
ret["provider_institution"] = provider_node.findtext(_t('institution'))

# get the chief complaint
ret["chief_complaint"] = doc_etree.findtext(_t('chiefComplaint'))

# get the content of the note
ret["content"] = doc_etree.findtext(_t('content'))

return ret

BaseTransform.to_sdmj(doc_etree)
Transform an etree into a string of valid Simple Data Model JSON.

Subclasses should implement this method, which takes an lxml.etree._ElementTree (the result of calling etree.parse()), and returns a string in valid SDMJ format.

As an example, here’s how you might implement this for a transform from our Simple Clinical Note Schema to our Simple Clinical Note Data Model. Note the reuse of the _get_data() function from above:

```python
from indivo.document_processing import BaseTransform

SDMJ_TEMPLATE = '''
{
"__modelname__": "SimpleClinicalNote",
"date_of_visit": "$(date_of_visit)s",
"finalized_at": "$(finalized_at)s",
"visit_type": "$(visit_type)s",
"visit_type_type": "$(visit_type_type)s",
"visit_type_value": "$(visit_type_value)s",
"visit_type_abbrev": "$(visit_type_abbrev)s",
"visit_location": "$(visit_location)s",
"specialty": "$(specialty)s",
"specialty_type": "$(specialty_type)s",
"specialty_value": "$(specialty_value)s",
"specialty_abbrev": "$(specialty_abbrev)s",
"signed_at": "$(signed_at)s",
"provider_name": "$(provider_name)s",
"provider_institution": "$(provider_institution)s",
"chief_complaint": "$(chief_complaint)s",
"content": "$(content)s"
}
'''

class Transform(BaseTransform):
```
def to_sdmj(self, doc_etree):
    args = self._get_data(doc_etree)
    return SDMJ_TEMPLATE % args

BaseTransform.to_sdmx(doc_etree)

Transform an etree into a string of valid Simple Data Model XML.

Subclasses should implement this method, which takes an lxml.etree._ElementTree (the result of calling etree.parse()), and returns another lxml.etree._ElementTree instance representing an XML document in valid SDMX format.

As an example, here’s how you might implement this for a transform from our Simple Clinical Note Schema to our Simple Clinical Note Data Model. Note the reuse of the _get_data() function from above:

```python
from indivo.document_processing import BaseTransform
from lxml import etree
from StringIO import StringIO

SDMX_TEMPLATE = '''
<Models>
    <Model name="SimpleClinicalNote">
        <Field name="date_of_visit">%(date_of_visit)s</Field>
        <Field name="finalized_at">%(finalized_at)s</Field>
        <Field name="visit_type">%(visit_type)s</Field>
        <Field name="visit_type_type">%(visit_type_type)s</Field>
        <Field name="visit_type_value">%(visit_type_value)s</Field>
        <Field name="visit_type_abbrev">%(visit_type_abbrev)s</Field>
        <Field name="visit_location">%(visit_location)s</Field>
        <Field name="specialty">%(specialty)s</Field>
        <Field name="specialty_type">%(specialty_type)s</Field>
        <Field name="specialty_value">%(specialty_value)s</Field>
        <Field name="specialty_abbrev">%(specialty_abbrev)s</Field>
        <Field name="signed_at">%(signed_at)s</Field>
        <Field name="provider_name">%(provider_name)s</Field>
        <Field name="provider_institution">%(provider_institution)s</Field>
        <Field name="chief_complaint">%(chief_complaint)s</Field>
        <Field name="content">%(content)s</Field>
    </Model>
</Models>
'''

class Transform(BaseTransform):
    def to_sdmx(self, doc_etree):
        args = self._get_data(doc_etree)
        return etree.parse(StringIO(SDXM_TEMPLATE % args))
```

### 5.24.4 Adding Custom Transforms to Indivo

Associating a new transform with an Indivo-supported schema is simple:

- Write your transform, as either an XSLT or a Python module, as described above.
- Drop the file containing your transform (transform.xslt or transform.py: make sure to name the file ‘transform’) into the directory containing the schema. See Adding Custom Schemas to Indivo for more details.
- Make sure to restart Indivo after moving transform files around, or the changes won’t take effect.
5.25 Unimplemented API Calls

Todo

The API Call ‘GET /carenets/{0}/apps/{1}/permissions’ is not yet implemented.

(The original entry is located in /var/build/user_builds/indivo-x/checkouts/2.0/doc/sphinx/source/api-reference.rst, line 1401.)

Todo

The API Call ‘GET /codes/systems/’ is not yet implemented.

(The original entry is located in /var/build/user_builds/indivo-x/checkouts/2.0/doc/sphinx/source/api-reference.rst, line 2049.)

Todo

The API Call ‘POST /records/{0}/documents/{1}/carenets/{2}/autoshare-revert’ is not yet implemented.

(The original entry is located in /var/build/user_builds/indivo-x/checkouts/2.0/doc/sphinx/source/api-reference.rst, line 3820.)

Todo

The API Call ‘PUT /records/{0}/documents/{1}/meta’ is not yet implemented.

(The original entry is located in /var/build/user_builds/indivo-x/checkouts/2.0/doc/sphinx/source/api-reference.rst, line 3921.)
6.1 Building an Indivo App using Python

This is a walkthrough on writing an Indivo X application that is embedded within Indivo X. In this example, the application will help to manage the user’s medical problems (i.e. arthritis, hypertension, ...)

6.1.1 Getting Started

An Indivo App is just a web site that:

• presents a web interface to the user, framed within the Indivo user interface
• connects to Indivo X on the backend to fetch and store data

The Indivo API defines how to call the Indivo X server on the backend.

This document explains the details of how to build an Indivo app, using Python. Of course, any other programming language / web platform can be used following the same principles.

Note: As of Indivo X Beta 1, this problems app is fully integrated into the Indivo UI Server, for ease of deployment. This does not change the fact that you can and should build additional apps as separate servers.

Scope

Our sample application will help track a user’s medical problem list, using coded-value lookups to fill in the problem name.

General Architecture and User Flow

A user will add the Indivo Problems application to their record on demand, or it may be added for them by an administrative application. User Authentication is entirely performed via Indivo.

Indivo Problems will store all of its data in Indivo X, using the Problem data model. Notes added to the problem will be stored as standard annotations. Thus, Indivo Problems does not require anything other than application logic: no database, no authentication mechanism, just HTML serving and access to the Indivo X API.
6.1.2 Authentication

The steps involved in adding an application to an Indivo record are:

- the user navigates to the App Settings pane in Indivo
- Indivo lists the available applications, including Indivo Problems, which the user selects and enables.
- Indivo opens up an IFRAME onto the app’s index URL, with either record_id or carenet_id as a URL query parameter.
- The app begins an oAuth authorization protocol with the Indivo backend server.
- The IFRAME is redirected to the Indivo authorization screen
- when the user approves the app, the IFRAME is redirected to the app’s oauth_callback URL, which can now complete the oAuth process and access Indivo.

Thus, our application needs:

- an index URL:
  
  /start_auth?record_id={record_id}&carenet_id={carenet_id}

  **Note:** Only one of those two params will be filled in.

- an oauth_callback URL that complies with the oAuth protocol (v1.0a):
  
  /after_auth?oauth_token={oauth_token}&oauth_verifier={oauth_verifier}

**IMPORTANT:** the Indivo system is split into a UI server, which presents the HTML user interface, and a BACKEND server, which communicates only in XML. The oAuth process involves both servers: oAuth calls are made against the BACKEND server, while the browser is redirected to the UI server.

**Understanding Record-Based vs. Carenet-Based**

When an app is invoked in Indivo, it is either invoked at the record level, or at the carenet level. A record-level app launch means that the app is launched by the record-owner, and can access all of that user’s data. A carenet-level app launch means that the app is launched by a guest of the record-owner, and can see only limited data within that specific carenet. In general record-level apps display additional logic that carenet-level apps do not, e.g. the sharing controls.

**Authentication Logic**

Let’s build the logic for these two entry points.

**Start**

We start with a python function declaration:

```python
def start_auth(request):
    """
    begin the oAuth protocol with the server
    """
```

and now we need access to Indivo to get our oAuth request token from the backend server:
server_params = {"api_base": settings.INDIVO_SERVER_LOCATION,
                 "authorization_base": settings.INDIVO_UI_SERVER_BASE}
consumer_params = settings.INDIVO_SERVER_OAUTH
client = IndivoClient(server_params, consumer_params)

Note how we’re pulling the credentials from the Django settings, which are stored in settings.py Also, in the actual codebase, we’ve modularized this to a get_indivo_client function.

Next, we check to see if we were passed a record_id or carenet_id parameter, which is what happens when Indivo opens up an IFRAME onto our start URL, since it knows exactly what record (or carenet) is currently being accessed. We use this record_id or carenet_id to set up our OAuth parameters, and then get ourselves a request token:

```python
# do we have a record_id or carenet_id?
record_id = request.GET.get('record_id', None)
carenet_id = request.GET.get('carenet_id', None)

# prepare request token parameters
params = {'oauth_callback': 'oob'}
if record_id:
    params['indivo_record_id'] = record_id
if carenet_id:
    params['indivo_carenet_id'] = carenet_id

# request a request token
request_token = client.fetch_request_token(params)
```

Now that we have this request token, it’s time to store it in the web session for later and send the user to Indivo for authorization:

```python
# store the request token in the session for when we return from auth
request.session['request_token'] = request_token

# redirect to the UI server
return HttpResponseRedirect(client.auth_redirect_url)
```

The redirect is now to the UI server, which is different from the backend server (the client takes care of this detail for you, since you simply ask for the auth_redirect_url)

And that’s it, we’re finished with half of the code needed to connect an app with Indivo X for authentication and medical-record connectivity!

Post Auth

Once the user has approved the application for addition, Indivo X will redirect the user to the oauth_callback URL at our Problems App web server, and now it’s time for us to complete the authentication process by converting our request token into an access token. We start with a new Python function:

```python
def after_auth(request):
    
    
    after Indivo authorization, exchange the request token for an access token and store it in the web
```

Then, we retrieve the request token we stored in the session, as well as the token string and oauth verifier we receive as URL parameters:

```python
# get the token and verifier from the URL parameters
oauth_token, oauth_verifier = request.GET['oauth_token'], request.GET['oauth_verifier']
```
We quickly check that the token in the URL parameter matches the web session, just to be extra safe:

```python
if token_in_session['oauth_token'] != oauth_token:
    return HttpResponse("uh oh bad token")
```

Then we connect to Indivo using the consumer secret but also the request-token details to exchange the request token for an access token:

```python
# get the indivo client and use the request token as the token for the exchange
server_params = {
    "api_base": settings.INDIVO_SERVER_LOCATION,
    "authorization_base": settings.INDIVO_UI_SERVER_BASE
}
consumer_params = settings.INDIVO_SERVER_OAUTH
client = IndivoClient(server_params, consumer_params)
client.update_token(token_in_session)
access_token = client.exchange_token(oauth_verifier)
```

Once again, in the actual code, we’ve modularized the client creation to the `get_indivo_client` function.

And that’s it, we’re fully connected! We now store the access token details in the web session for later use, and redirect to the app’s homepage:

```python
# store stuff in the session
request.session['access_token'] = access_token

# depending on whether we get a record or carenet id back.
if access_token.has_key('xoauth_indivo_record_id'):
    request.session['record_id'] = access_token['xoauth_indivo_record_id']
else:
    request.session['carenet_id'] = access_token['xoauth_indivo_carenet_id']

# go to list of problems
return HttpResponseRedirect("/")
```

Notice how the access token came back with an extra parameter that indicates the identifier of the Indivo record we just managed to bind, or of the carenet.

**URL handlers**

We build URL handlers in Django’s `urls.py`:

```python
from views import start_auth, after_auth
urlpatterns = patterns(' ',
    (r'^start_auth', start_auth),
    (r'^after_auth', after_auth),
)
```

### 6.1.3 Recording and Displaying Problems

The rest of the application is a standard web app that displays a list of problems and lets the user add a new one. The generic web components are best explained by the existing Django documentation. Here, we cover briefly the Indivo-specific touchpoints.
Indivo X Documentation, Release 2.0.0

Getting information from Indivo

Every call to the Indivo Problem List app requires information from Indivo. Thus, in every call, it is useful to set up
the client front-end to Indivo as:

server_params = {"api_base": settings.INDIVO_SERVER_LOCATION,
                 "authorization_base": settings.INDIVO_UI_SERVER_BASE}
consumer_params = settings.INDIVO_SERVER_OAUTH
client = IndivoClient(server_params, consumer_params)
client.update_token(request.session['access_token'])

In the Indivo Problem List code, this is packaged as get_indivo_client in the utils.py file.

Reading a list of Problems

Though each problem is its own Indivo document, problems might come from a CCR, from a list of problems in
another schema, etc... Thus, it is always best to access Problems through Problems Report when listing problems,
which will list all of the reports processed from all input documents.

The call is slightly different depending on whether this is a record or carenet (eventually, Indivo may provide a single
API call to make this easier, but for now we must differentiate):

client = get_indivo_client(request)

if request.session.has_key('record_id'):
  record_id = request.session['record_id']

  # Note that we're asking for our response data in JSON form: we could also get it as XML or RDF
  resp, content = client.generic_list(record_id=record_id, data_model="Problem", body={'response_type':'application/json'})
  if resp['status'] != '200':
    # TODO: handle errors
    raise Exception("Error reading problems: %s"%content)
  probs = simplejson.loads(content)

else:
  carenet_id = request.session['carenet_id']
  # Read problems from the carenet: This also returns JSON, which is the default return type for data
  resp, content = client.carenet_generic_list(carenet_id=carenet_id, data_model="Problem")
  if resp['status'] != '200':
    # TODO: handle errors
    raise Exception("Error reading problems from carenet: %s"%content)
  probs = simplejson.loads(content)

Notice that we’ve used the simplejson library to parse our JSON return data. It is now available to us in the probs
variable, as a python array that will look like:

[{
  "__modelname__": "Problem",
  "__documentid__": "12345",
  "startDate": "2009-05-16T12:00:00Z",
  "endDate": "2009-05-16T16:00:00Z",
  "name_title": "Backache (finding)",
  "name_system": "http://purl.bioontology.org/ontology/SNOMEDCT/",
  "name_identifier": "161891005"
},
... More Problems ...]
Creating a Document

To create a document, one must first put together the necessary XML. The way we do this in our sample application is to use Django’s templating system to interpolate values into the XML template for Problem:

```python
# get the variables and create a problem XML
params = {'coding_system': 'http://purl.bioontology.org/ontology/SNOMEDCT/',
          'date_onset': request.POST['date_onset'],
          'date_resolution': request.POST['date_resolution'],
          'code_fullname': request.POST['code_fullname'],
          'code': request.POST['code'],
          'comments': request.POST['comments']}
problem_xml = render_raw('problem', params, type='xml')

Then, we submit this as a new document:

```python
resp, content = client.document_create(record_id=request.session['record_id'], body=problem_xml,
                                        content_type='application/xml')
``` 

if resp['status'] != '200':
    # TODO: handle errors
    raise Exception("Error creating new problem: %s"%content)

Coded Values

In the course of creating a document, one needs to access coded values, for example SNOMED codes. Indivo makes coded values available via its API, e.g:

```python
resp, content = client.coding_system_query(system_short_name='snomed', body={'q':query})
``` 

which will return a JSON list of codes, each with properties `abbrev`, `code`, `physician_value`, `umls_code`, and `consumer_value`.

In our sample application, we take this return value and format it for the jQuery Autocomplete Plugin:

```python
query = request.GET['query']
resp, content = client.coding_system_query(system_short_name='snomed', body={'q':query})
if resp['status'] != '200':
    # TODO: handle errors
    raise Exception("Error getting coding systems data: %s"%content)
codes = simplejson.loads(content)
formatted_codes = {'query': query, 'suggestions': [c['consumer_value'] for c in codes], 'data': codes}
return HttpResponse(simplejson.dumps(formatted_codes), mimetype="text/plain")

Reading a single Document

From the report, we can get the `document_id` from which each problem is extracted. Using this `document_id`, it’s easy to get the original document itself. In this case, the document won’t contain any extra information form what was found inside the report, but oftentimes the document will contain more detail or other contextual data.

Again, we must be conscious of whether this is within a record or carenet:

```python
record_id = request.session.get('record_id', None)
if record_id:
    resp, content = client.record_document(record_id=record_id, document_id=problem_id)
    if resp['status'] != '200':
```
# TODO: handle errors
raise Exception("Error fetching document: \$s\$%content")
doc_xml = content

else:
carenet_id = request.session[‘carenet_id’]
# read the document
resp, content = client.carenet_document(carenet_id=carenet_id, document_id=problem_id)
if resp[‘status’] != ‘200’:
    # TODO: handle errors
    raise Exception("Error fetching document from carenet: \$s\$%content")
doc_xml = content

Notifying the Record

Sometimes, a PHA needs to notify a record of some action:

client.record_notify(record_id=request.session[‘record_id’],
    body=(‘content’:‘a new problem has been added to your problem list’))

6.1.4 Adding UI Widgets

Indivo X, as of alpha 2, supports UI widgets that an app can easily integrate into its interface. The first such widget is “Sharing and Audit”, which lets a user modify the sharing preferences and quickly view the audit log for a particular document. This sharing widget should really only be displayed when the app is record-level.

Getting SURL Credentials

To invoke a widget, an app must first generate SURL credentials, i.e. credentials that will allow it to generate Signed URL. Signed URLs ensure that only authorized apps can embed a specific widget. Fortunately, the Indivo client provides a simple built-in method for generating these SURL credentials:
surl_credentials = client.get_surl_credentials()

Setting up the JavaScript

Once SURL credentials have been generated, it’s time to load the widget JavaScript and initialize it. This is done in the HTML template:

<script src="{INDIVO_UI_SERVER_BASE}\lib/widgets.js"></script>

then:

<script>
    Indivo.setup('{INDIVO_UI_SERVER_BASE}');
</script>

and:

<script>
    Indivo.Auth.setToken("{surl_credentials.token}","{surl_credentials.secret}");
</script>
Adding the Widget

Finally, it’s time to add the widget:

{% if record_id %}
<script>
</script>
{% endif %}

Note how this widget is only added if there is a record_id, since a carenet-level app should not display the sharing widget. And, in fact, if it tried, it wouldn’t know the record_id needed, and if it guessed it correctly it would not have the right permissions to do so.

6.2 Installation Instructions for OSX Lion

6.2.1 Requirements Overview

- OS X 10.7 & 10.8
- PostgreSQL 9.0.4
- Python 2.7.1
- Django 1.3.1
- lxml 2.3.4
- psycopg2 2.4.5
- git 1.7.4.4

Optional:

- Apache 2.2.18 (Unix)
- mod_wsgi 3.3

6.2.2 Prerequisites

Command Line Tools

If you have Xcode installed, make sure the command line tools are also installed (in Xcode > Preferences > Downloads > Components > Command Line Tools). You can download Xcode for free from the Mac App Store, but it will cost you several GB of disk space. Alternatively, you can register a free developer account with Apple and download an installer for just the command line tools.

You should now have the compilers and git installed.

PostgreSQL

It’s easiest to use the Mac installer because it also sets up the postgres user which is needed.

- Download the latest installer (I used 9.1.4) and run it. You can keep most default settings:
  - Install into /Library/PostgreSQL/9.1
  - Port 5432
– Remember your password!
– Locale en_US.UTF-8 (any .UTF-8 will do)

• Your PostgreSQL Server should now be running. You can restart Postgres with:
  
  ```sh
  sudo launchctl stop com.edb.launchd.postgresql-9.1 (it will then automatically restart)
  ```

**Make PostgreSQL more secure; only needed when using Postgres < 9.1!**

• Login as postgres if you’re not already:
  
  ```sh
  sudo su - postgres
  ```
  (We should now be in /Library/PostgreSQL/9.0)

• Make the changes to `data/pg_hba.conf`: Close to the bottom of the file, change the lines that end with “ident” or “trust” into “md5”, for example:
  
  ```
  local   all    all    ident
  to:
  local   all    all    md5
  ```

**Python: lxml**

Instructions taken from: [http://lxml.de/build.html#building-lxml-on-macos-x](http://lxml.de/build.html#building-lxml-on-macos-x)

• Download: [http://pypi.python.org/pypi/lxml/2.3.4#downloads](http://pypi.python.org/pypi/lxml/2.3.4#downloads) (download the source tarball, double click to unarchive)

• Build & Install:
  
  ```sh
  cd lxml-2.3.4
  python setup.py build --static-deps
  sudo python setup.py install
  ```

**Python: psycopg2**

• Download: [http://www.psycopg.org/psycopg/tarballs/PSYCOPG-2-4/psycopg2-2.4.5.tar.gz](http://www.psycopg.org/psycopg/tarballs/PSYCOPG-2-4/psycopg2-2.4.5.tar.gz) (download the source, double click to unarchive)

• Build & Install:
  
  ```sh
  cd psycopg2-2.4.5
  python setup.py build
  sudo python setup.py install
  ```

**Python: Markdown support**

```sh
sudo easy_install Markdown
```

**Python: RDF support**

6.2. Installation Instructions for OSX Lion
sudo easy_install rdflib

**Django**

- **Download:** https://www.djangoproject.com/download/ (download the source tarball, version 1.3 is currently needed!)
- **Install:**
  
  sudo python setup.py install

**Django South DB migration library**

sudo easy_install South

### 6.2.3 Indivo Setup

- It’s easiest if you put both indivo_server and indivo_ui_server into the same directory (I used /Library/Indivo), the commands below will do just that

- If you clone the Indivo code from github, don’t forget to init the submodules:

  ```
  cd /Library/Indivo/
  git clone git@github.com:chb/indivo_server.git
  cd indivo_server/
  git submodule init
  git submodule update
  cd ..
  git clone git@github.com:chb/indivo_ui_server.git
  cd indivo_ui_server/
  git submodule init
  git submodule update
  ```

- Submodules may stall, you can clone them by hand (these three did stall for me):

  ```
  cd indivo_ui_server/ui/jmvc/
  git clone git@github.com:jupiterjs/funcunit
  git clone git@github.com:jupiterjs/jquerymx jquery
  git clone git@github.com:jupiterjs/steal
  cd ..
  git submodule update
  ```

**Create the Indivo postgres user**

This will create a PostgreSQL user named indivo and any password. Be sure to remember the password as you will need to put it into the Indivo settings later!:

```
sudo su - postgres
createuser --superuser indivo
psql postgres
postgres=# \password indivo
postgres=# \q
```
Follow the instructions to configure Indivo

Server Config

UI Server Config

Optional for Apache: mod_wsgi

Only needed if you want to run Indivo on top of Apache

- Check: Run `httpd -M` to check whether you have it already installed
- If not, download: [http://code.google.com/p/modwsgi/downloads/detail?name=mod_wsgi-3.3.tar.gz](http://code.google.com/p/modwsgi/downloads/detail?name=mod_wsgi-3.3.tar.gz) (There is an installer ready made for OS X 10.6, but we’re using the tarball here)
- Build & Install:
  ```
  ./configure
  make
  sudo make install
  ```
- Setup: To have Apache load the module, add this line to `/etc/apache2/httpd.conf`:
  ```
  LoadModule wsgi_module libexec/apache2/mod_wsgi.so
  ```
- Follow the Wiki instructions to setup Apache. For OS X, there are minor deviations of the procedure. Two hints:
  - The virtual hosts config is in: `/etc/apache2/extra/httpd-vhosts.conf`. Uncomment the inclusion command for this file in the main `httpd.conf`
  - Restart Apache with:
    ```
    sudo apachectl graceful
    ```
- Some permission settings, make sure Apache has access to these files/directories:
  ```
  indivo_server/indivo.log  indivo_ui_server/settings
  indivo_ui_server/indivo_client_py
  ```

6.3 Installation Instructions for RHEL 5

Contributed by Timothy Kutz and Matthew Ellison.

6.3.1 Pre-req Installs

- RHEL 5.6
- Oracle 10g Client – installed to `/data2/app/oracle`
- Python 2.6.4 – installed to `/usr/bin/python2.6` with libs at `/usr/lib/python2.6`
- Django 1.2.5 – installed to python2.6 location
- `cx_Oracle-5.0.4-10g-py26-1.x86_c6.rpm`
- Apache 2.2.18 – installed to `/data2/app/apache22`
- Mod_wsgi 3.3 for Python 2.6.4 – installed to apache modules
6.3.2 Install Indivo

- Indivo_server codebase – installed to /data2/app/apphome/indivo: Ensure this location is readable by the user Apache will run under (usually daemon)
- Using Virtualenv tool (http://www.virtualenv.org/en/latest/index.html), create python26 environment in /data2/app/python/python26:

  ```
  $ Python virtualenv.py /data2/app/python/python26
  ```

- Ensure the links created here point correctly to the python 2.6 locations. If it points to 2.4, something is wrong, possibly the version of python invoked by the user running the tool.
- Ensure this environment is permissioned 755 (public readable/executable).

6.3.3 Apache Config

- Add to $APACHE_HOME/bin/envvars:

  ```
  ORACLE_HOME="/data2/app/oracle"
  export ORACLE_HOME
  
  LD_LIBRARY_PATH="/data2/app/apache22/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH"
  export LD_LIBRARY_PATH
  
  PATH="$ORACLE_HOME/bin:$PATH"
  export PATH
  ```

- Add to $APACHE_HOME/conf/httpd.conf:

  ```
  (in modules section): 
  LoadModule wsgi_module modules/mod_wsgi.so
  
  (at end of file): 
  <IfModule wsgi_module>
      DocumentRoot /data2/app/apphome/indivo/indivo_server
      Alias /static/ /data2/app/apphome/indivo/indivo_server/static/
      EnableMMAP On
      EnableSendfile On
      LogLevel warn
      
      <Directory /data2/app/apphome/indivo/indivo_server>
          Order deny,allow
          Allow from all
      </Directory>
      WSGIPythonHome /data2/app/python/python-26/
      WSGIApplicationGroup %{GLOBAL}
      WSGIScriptAlias / /data2/app/apphome/indivo/indivo_server/django.wsgi
      WSGIPassAuthorization On
  </IfModule>
  ```

6.3.4 Django/Indivo Config

- Modify settings.py (Only modified lines shown, in order):
# base URL for the app
APP_HOME = '/data2/app/apphome/indivo/indivo_server'

# URL prefix
SITE_URL_PREFIX = "http://ndvodmo.tch.harvard.edu"

...  

# Oracle DB settings

#NOTE: DATABASE_NAME must be under 255 chars, so all whitespace has been removed below.
DATABASE_ENGINE = 'django.db.backends.oracle'
DATABASE_NAME = '(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=HOST1)(PORT=1550))(ADDRESS=(PROTOCOL=TCP)(HOST=HOST2)(PORT=1550))(LOAD_BALANCE=yes)(CONNECT_DATA=(SERVER=DEDICATED(
	SERVICE_NAME=CHRACTST)
	(FAILOVER_MODE=(TYPE=SELECT)(METHOD=BASIC)(RETRIES=180)(DELAY=5))))')

DATABASE_USER = 'USERNAME'
DATABASE_PASSWORD = 'PASS'
DATABASE_HOST = ''
DATABASE_PORT = ''

# logging
#import logging
#logging.basicConfig(level = logging.DEBUG, format = '%(asctime)s %(levelname)s%(message)s',
#                   filename = '/data2/app/indivo/logs/indivo.log', filemode = 'w')
#

6.3.5 Set Up Oracle Instance

• The Oracle instance(s) serving Indivo must match the connection string in settings.py under
  DATABASE_NAME. This will involve setting up a tablescape for Indivo (in the above example CHRACTST),
  and user access.

6.4 Installation Instructions for Ubuntu

6.4.1 Defaults

This document sets up the Indivo backend server and Indivo UI server on the same machine. The backend server runs
on port 8000, the UI server on port 80 by default. This is configurable by changing the appropriate port numbers in
the instructions below.

6.4.2 Pre-Requisites

This documents the installation of a complete Indivo X server and user interface. For concreteness, we show all of the
exact instructions needed when installing on Ubuntu Linux 11.04 (Natty) or 11.10 (Oneiric). We welcome variants of
these explicit instructions from folks installing on other operating systems, we will be happy to post them alongside
these instructions.

Note: We recommend you do this by sudo’ing from a non-root user. If you would like to do this as root make sure
you create at least one non-root user with:

useradd -m {USER}

otherwise the default locale will not be set. This issue is most common on a new OS build.

You will need the following for Indivo:

• Recent Linux installation (Kernel 2.6+)
• Python 2.6+ (NOT 2.5 or below) with package lxml:
  
  apt-get install python-lxml

• Django 1.2 or 1.3 (1.1 is NO LONGER SUPPORTED)
  
  From source:
  
  wget http://www.djangoproject.com/download/1.3.1/tarball/
  tar xzvf Django-1.3.1.tar.gz
  cd Django-1.3.1
  sudo python setup.py install
  cd ..
  rm -rf Django-1.3.1 Django-1.3.1.tar.gz

  Using a package manager (Ubuntu 10.10 or later):
  
  apt-get install python-django

• Apache 2 (for production) with module mod_wsgi (mod_ssl for HTTPS is strongly recommended for production, but we won’t cover its installation here.):

  apt-get install apache2-mpm-prefork

  Note: For Lucid, you may need to do an apt-get update before making the following command:
  
  apt-get install libapache2-mod-wsgi

• easy_install for Python:

  apt-get install python-setuptools

• the Django South DB migration library:

  easy_install South

• Python Markdown support:

  easy_install Markdown

• RDF support:

  easy_install rdflib

6.4.3 Database Install

Postgres

Install

• PostgreSQL 8.3+ (8.4 recommended and the default on Ubuntu 10.10):

  apt-get install postgresql

• The psycopg python binding for postgres.

  For Ubuntu 11.04 and below, simply install with:
apt-get install python-psycopg2

Recent versions of psycopg2 (2.4.2+) are incompatible with Django. If your default version is 2.4.2 or greater (true as of Ubuntu 11.10), install version 2.4.1 instead, from http://initd.org/psycopg/tarballs/PSYCOPG-2-4/psycopg2-2.4.1.tar.gz (instructions for installation may be found at http://initd.org/psycopg/install/). If you’ve already installed a newer version of psycopg2, you can downgrade with:

```
pip install psycopg2==2.4.1
```

**Setup**

**Note:** You’ll have the easiest time naming your database indivo

There are two ways to authenticate to PostgreSQL: use your Unix credentials, or use a separate username and password. We strongly recommend the latter, and our instructions are tailored appropriately. If you know how to use PostgreSQL and want to use Unix-logins, go for it, just remember that when you use Apache, it will usually try to log in using its username, www-data.

In /etc/postgresql/8.4/main/pg_hba.conf, find the line that reads:

```
local   all   all   ident
```

This should be the second uncommented line in your default config. Change it to:

```
local   all   all   md5
```

You will need to restart PostgreSQL:

```
service postgresql-8.4 restart
```

or, on postgres 9+:

```
service postgresql restart
```

Create a PostgreSQL user for your Indivo service, e.g. “indivo” and setup a password:

```
su - postgres
createuser --superuser indivo
psql
postgres=# \password indivo
postgres=# \q
logout
```

Create the Database and make the Indivo user its owner:

```
createdb -U indivo -O indivo indivo
```

**More Information**

See the Django PostgreSQL notes

6.4. Installation Instructions for Ubuntu
MySQL

Install

- The Mysql server (only tested with v5.0+):
  ```
  sudo apt-get install mysql-server
  ```
  When prompted, enter a password for the root user.

- The MySQLdb python binding (v1.2.1p2 or greater required for django):
  ```
  sudo apt-get install python-mysqldb
  ```

Setup

- Change the default storage engine to InnoDB, which supports transactions. In /etc/mysql/my.cnf, find the line reading [mysqld]. Directly underneath that line, add:
  ```
  default-storage-engine = innodb
default-character-set = utf8
  ```
  Then restart mysql with:
  ```
  sudo service mysql restart
  ```

- Create the Indivo User by logging in (using the password you set up for the root user during install):
  ```
  mysql -uroot -p
  ```
  And then running the following commands:
  ```
  > CREATE USER 'indivo'@'%' IDENTIFIED BY 'YOURPASSWORD'; # Replace YOURPASSWORD with a password
  > GRANT ALL PRIVILEGES ON *.* TO 'indivo'@'%' WITH GRANT OPTION;
  > exit
  ```

- Create the Indivo Database:
  ```
  mysqladmin -u indivo -p create indivo
  ```
  Authenticating with the password you set up for the indivo user.

Idiosyncracies

Date formatting doesn’t work quite the same as it does on the other backends. Specifically:

- “Week of the Year” (00-53), which normally counts weeks as increments of 7 days starting at Jan 01, in mysql counts week 0 as anything before the first Sunday of the year, and after that counts weeks in increments of 7 days, starting on Sunday.

- “Day of the Week”, which is normally indexed from 1 to 7, starting on Sunday, on mysql is indexed from 0 to 6, starting on Sunday.

More Information

See the Django MySQL notes
Oracle

Install

If you do not have a supported installation of Oracle already, the odds are good that you shouldn’t be running Indivo on Oracle. Also, Oracle doesn’t play nicely with Debian Linux, so you also probably shouldn’t be setting it up on Ubuntu. These installation instructions assume that you have a running instance of Oracle on another machine, and describe how to connect to it from an Indivo instance running on Ubuntu.

You’ll need an installation of Oracle against which to bind the Python drivers. You can use Oracle XE (express edition), which is free and based on Oracle 11i. We used the following installation instructions.

Note: These instructions only work for 32-bit Linux. For 64-bit versions, Oracle doesn’t offer a solution.

Get the Python Oracle driver, cx_Oracle, with installation instructions explained here.

Setup

Set up your Oracle user on the remote system. From the Django docs, you’ll need to insure that you have access to your Oracle instance as a user with the following privileges:

• CREATE TABLE
• CREATE SEQUENCE
• CREATE PROCEDURE
• CREATE TRIGGER

To run Indivo’s test suite, the user needs these additional privileges:

• CREATE USER
• DROP USER
• CREATE TABLESPACE
• DROP TABLESPACE
• CONNECT WITH ADMIN OPTION
• RESOURCE WITH ADMIN OPTION

Make sure your environment variables are set properly as described in the install instructions for cx_Oracle. Importantly:

• set ORACLE_HOME to the home directory for oracle, /usr/lib/oracle/xe/app/oracle/product/10.2.0/server, by default
• set LD_LIBRARY_PATH to $ORACLE_HOME/lib
• add $ORACLE_HOME/bin to your $PATH variable.

If you intend on running Indivo on Apache, the Apache user will also need access to these environment variables. You can set this up by editing /etc/apache2/envvars and adding the above variable declarations.

Test that cx_Oracle has been installed. If the following command exits silently, your setup is correct:

python -c "import cx_Oracle"
More Information

See the Django Oracle notes

6.4.4 Indivo Server

Get the Code

From A Packaged Release

• Download the latest release of Indivo X from our tags page and untar into indivo_server/. Do not change this directory name—it will break the django settings file.

From Github

• From the commandline, run:
  cd /desired/install/directory
  git clone --recursive git://github.com/chb/indivo_server.git

• If you want to run the bleeding edge version of Indivo, you’re done. If you want to use an official release, you can list releases with:
  git tag -n1

  and checkout your desired release and update its submodules with:

  git checkout {TAGNAME}
  git submodule update

where tagname might be (i.e., for version 2.0) v2.0.0.

Configuration

Copy settings.py.default to settings.py, and open it up. Make sure to look at the ‘Required Setup’ settings, and examine ‘Advanced Setup’ if you are interested. As an absolute minimum, update the following:

• set SECRET_KEY to a unique value, and don’t share it with anybody

• set APP_HOME to the complete path to the location where you’ve installed indivo_server, e.g. /web/indivo_server

• set SITE_URL_PREFIX to the URL where your server is running, including port number e.g. https://pchr.acme.com:8443

• Database Settings: Edit the ‘default’ database under DATABASES, and:
  – set ENGINE to the database backend you are using, prefixed by ‘django.db.backends.’. Available options are ‘postgresql_psycopg2’, ‘mysql’, and ‘oracle’.
  
  – set NAME to the name you would like to use for your database. If you followed the database setup instructions above, you should leave this as ‘indivo’.
  
  – set USER to the username you chose, in this documentation indivo, and set PASSWORD accordingly.
  
  – If your database is located on another machine, set HOST and PORT appropriately.
– If you are running Oracle, see https://docs.djangoproject.com/en/1.2/ref/databases/#id11 for how to configure the database settings.

• set the SEND_MAIL parameter to True or False depending on whether you want emails actually being sent.

• set the EMAIL_* parameters appropriately for sending out emails.

• Under utils/copy indivo_data.xml.default to indivo_data.xml and edit accordingly.

Note: Make sure to do this step before resetting the database

Reseting the Database

• On postgres or mysql from your base install directory:
  
  python utils/reset.py

• On other database backends, we don’t yet have reset scripts. You can reset Indivo by:
  
  – Flushing the database:
    
    python manage.py flush
  
  – Telling South that the database is actually in the correct state after migration:
    
    python manage.py migrate --fake
  
  – Importing the initial Indivo data:
    
    python utils/importer.py -v
  
  – Loading Coding Systems (optional):
    
    python load_codingsystems.py

You must run reset.py or utils/importer.py before the accounts and applications you set up in indivo_data.xml exist.

Coding Systems

TODO Indivo uses SNOMED CT for problem coding, HL7v3 for immunization coding, and LOINC for lab coding. Medication coding will likely use RxNorm. In most cases, the license on these coding systems does not allow us to redistribute these codes with Indivo. We don’t like this. We wish we had truly free coding systems for health. We’ve told the folks at the National Library of Medicine as much. But there’s not much we can do about this for now.

What we’ve done is make it easy for you to load coding systems into Indivo if you can get them independently. Get the HL7 v3 codes from hl7.org, get the SNOMED CT dataset from UMLS. You should see “|”-separated files. Examples of how these files are formatted can be found in codingsystems/data/sample. Once you’ve downloaded these files independently from the coding system agencies, copy them to:

• codingsystems/data/complete/SNOMEDCT_CORE_SUBSET_200911_utf8.txt
• codingsystems/data/complete/HL7_V3_VACCINES.txt
• codingsystems/data/complete/LOINCDB.txt

Once that’s done, assuming you’ve installed everything in /web/indivo_server, you can run:

utils/reset.sh.py -c
and have everything loaded properly.

See more info on codingsystems and where to find the data files here.

Database Cleanup

Each request made against Indivo Server generates some oauth-related data that is stored in the database for security reasons (for example, session tokens are stored whenever a user logs in, and OAuth Nonces are stored for every request). This data is only relevant for a certain duration (i.e., the length of a web session), after which point it becomes needless clutter in the database. In order to remove all such clutter, from APP_HOME run:

```
python manage.py cleanup_old_tokens
```

This command should be set up to run as a cron job, and should be run regularly to make sure the size of the database doesn’t get out of control (we recommend at least once a week, and more frequently for high traffic installations).

Testing Indivo Backend Server

Indivo uses the django-tests framework to provide some basic unit and API testing. If you want to make sure everything is setup properly before opening the server up to the network, running these tests is a good start. Django tests set up a clean test database for each run of the tests, so don’t worry about your installation being corrupted. To run the Indivo tests, in APP_HOME run:

```
python manage.py test indivo
```

6.4.5 Indivo UI Server

Get the Code

From a Packaged Release

- Download the latest release of Indivo X UI Server from our tags page and untar into indivo_ui_server/.

  **Note:** Do not change this directory name—it will break the django settings file.

From Github

- From the commandline, run:
  
  ```
  cd /desired/install/directory
  git clone --recursive git://github.com/chb/indivo_ui_server.git
  ```

- If you want to run the bleeding edge version of Indivo, you’re done. If you want to use an official release, you can list releases with:

  ```
  git tag -n1
  ```

  and checkout your desired release and update its submodules with:

  ```
  git checkout {TAGNAME}
  git submodule update
  ```

  where tagname might be (i.e., for Version 2.0) v2.0.0
Configuration

- Copy `settings.py.default` to `settings.py`, and update a few key parameters:
  - set `SERVER_ROOT_DIR` to the complete filesystem path to the location where you’ve installed `indivo_ui_server`, e.g., `/web/indivo_ui_server`, with no trailing slash.
  - set `INDIVO_UI_SERVER_BASE` to the URL at which your UI server will be accessible, e.g., `http://localhost`, with no trailing slash.
  - set `INDIVO_SERVER_LOCATION`, `CONSUMER_KEY`, `CONSUMER_SECRET` appropriately to match the Indivo Server’s location and chrome credentials (check `indivo_server/utils/indivo_data.xml` BEFORE you reset the database on the indivo_server end).
  - set `SECRET_KEY` to a unique value, and don’t share it with anybody.

6.4.6 Running Indivo

Django Development Servers

The Django development servers are easy to run at the prompt. The backend server can run on localhost in the configuration given above:

```
cd /web/indivo_server/
python manage.py runserver 8000
```

The UI server, if you want it accessible from another machine, needs to specify a hostname or IP address. If you want port 80, you need to be root of course:

```
cd /web/indivo_ui_server/
python manage.py runserver HOSTNAME:80
```

IMPORTANTLY, if you’ve installed Apache, you’ll need to turn it off to run your UI server from the prompt:

```
/etc/init.d/apache2 stop
```

Apache

Assuming you installed Indivo Server and UI in `/web`, the steps to getting Apache2 serving Indivo and its UI are:

- in `/etc/apache2/sites-available/default`, add:

  ```
  <VirtualHost *:8000>
  ServerAdmin YOU@localhost
  ServerName localhost
  DocumentRoot /web/indivo_server
  Alias /static/ /web/indivo_server/static/
  EnableMMAP On
  EnableSendfile On
  LogLevel warn
  
  <Directory /web/indivo_server>
  Order deny,allow
  Allow from all
  </Directory>
  ```
WSGIApplicationGroup %{GLOBAL}
WSGIScriptAlias /web/indivo_server/django.wsgi
WSGIPassAuthorization On
</VirtualHost>

<VirtualHost *:80>
  ServerAdmin YOU@localhost
  ServerName localhost
  DocumentRoot /web/indivo_ui_server
  Alias /static/ /web/indivo_ui_server/ui/static/
  EnableMMAP On
  EnableSendfile On
  LogLevel warn

  <Directory /web/indivo_ui_server>
    Order deny,allow
    Allow from all
  </Directory>

  WSGIDaemonProcess indivo_ui user=www-data group=www-data processes=1 maximum-requests=500 threads=10
  WSGIScriptAlias / /web/indivo_ui_server/django.wsgi
  WSGIPassAuthorization On
</VirtualHost>

In our experience, using WSGIProcessGroup directive with a specific group (not global), even when it matches the WSGIDaemonProcess group name (i.e. indivo_ui), can cause a permission issue with reading the Unix socket. We will continue to investigate this issue. However, due to incompatibilities between the lxml package and mod_wsgi, it is necessary to set Indivo Server to the global WSGIProcessGroup instead of running daemons.

- Make sure ports.conf has:
  NameVirtualHost *:80
  Listen 80
  Listen 8000

- Make sure that www-data (or whoever is in /etc/apache2/envvars) has access to indivo_server and indivo_ui_server AND can write to indivo_server/indivo.log and indivo_ui_server/sessions/*, including the sessions/ directory itself.

- Since you probably did a python manage.py syncdb, you almost certainly want to just remove the current indivo.log before you move ahead.

- Really, have you checked this www-data permission issue? This will be the cause of all your problems if you don’t check this carefully.

- Check your /etc/apache2/sites-enabled/000-default file again and make sure that your Alias /static/ lines match the above example exactly

- Restart Apache:
  service apache2 restart

6.4.7 What Next?

You should be able to log in and add the default apps. These apps are purposely limited in functionality. May the best apps win.
6.5 Upgrade Instructions

This article documents the necessary steps to upgrade Indivo to a newer version from an older one. In general, **Migration** refers to updating database schemas when moving from one version to another, presumably without losing any data in the process. As of the Beta 1 release, Indivo will provide scripts to migrate its database to the latest version. The rest of this page is concerned with other steps necessary to bring Indivo up to date with the latest release.

6.5.1 Get the new codebase

- Back up your old codebase: you wouldn’t want to lose all of your work!
- Option 1: if you are using the git repos:
  ```
  $ cd /path/to/indivo_server
  $ git pull origin master
  $ git checkout v1.0.0  # Or your desired version
  $ git submodule update
  
  $ cd /path/to/indivo_ui_server
  $ git pull origin master
  $ git checkout v1.0.0  # Or your desired version
  $ git submodule update
  ```
- Option 2: if you’d like to use the git repos, but aren’t already:
  ```
  $ cd /path/to/put/new/code
  $ git clone git://github.com/chb/indivo_server.git
  $ git clone git://github.com/chb/indivo_ui_server.git
  
  $ cd indivo_server
  $ git checkout v1.0.0  # Or your desired version
  $ git submodule init
  $ git submodule update
  
  $ cd ../indivo_ui_server
  $ git checkout v1.0.0  # Or your desired version
  $ git submodule init
  $ git submodule update
  ```
- Option 3: use a packaged release:
  ```
  $ cd /path/to/put/new/code
  $ wget https://github.com/downloads/chb/indivo_server/indivo_server-v1.0.0.tar.gz  # Or
  $ tar xzvf indivo_server-v1.0.0.tar.gz
  $ tar xzvf indivo_ui_server-v1.0.0.tar.gz
  $ rm indivo_*.tar.gz
  ```

6.5.2 Configure the New Codebase

**Indivo Server**

- Back up your old `settings.py` file, if you have one.
- Copy `settings.py.default` to `settings.py`, and merge your old changes into that file.
**Warning:** If you don’t move to the latest settings file, you might miss new settings or changes to settings that will break Indivo.

- Check your file permissions (remember, the user running Indivo must have write access to `indivo_server/indivo.log`).
- Regenerate the python test client, which might have changed to a new version:

  ```bash
  $ python indivo/tests/client/lib/create_api.py
  ```

  (This script does not output anything, so don’t be alarmed when it ends silently).

### Indivo UI Server

- Back up your old `settings.py` file, if you have one.
- Copy `settings.py.default` to `settings.py`, and merge your old changes into that file.

**Warning:** If you don’t move to the latest settings file, you might miss new settings or changes to settings that will break Indivo UI.

- Check your file permissions (remember, the user running Indivo must have write access to `indivo_ui_server/sessions/` and all of its contents).
- For versions < 2.0.0, regenerate the python client, which might have changed to a new version:

  ```bash
  $ python indivo_client_py/lib/create_api.py
  ```

  (This script does not output anything, so don’t be alarmed when it ends silently).

### 6.5.3 Update Your Database

**Migration:** If you have data you need to keep

First, **BACK UP YOUR DATABASE!!** Then, run the Indivo migration script. Depending on your version, the call might look slightly different.

**Note:** The package we rely on to handle Indivo data migrations, South, is still in alpha for the Oracle backend. If you are running Oracle, you may not be able to run the migration scripts below.

#### 2.0

Unfortunately, with the adoption of SMART data models, there is **no automatic migration path** from previous versions. The SMART data models are not too different in content from the previous Indivo ones, so it is possible to write an app for migration if you need to make the transition.

#### 1.0

For version 1.0, use the same migration script as previously, with the new target of ‘1.0’:

```bash
./migration_scripts/migrate_indivo.sh 1.0
```
Beta 3

As of Beta 3, the migration script now takes an additional argument: the version you wish to migrate to. If you have the Beta 3 indivo codebase or later, in the Indivo Home Directory (`indivo_server`, if you installed Indivo according to our instructions), run:

```
./migration_scripts/migrate_indivo.sh <target>
```

Acceptable targets are `beta1`, `beta2`, or `beta3`

Beta 2

Migrating Indivo is as simple as it gets. In the Indivo Home Directory (`indivo_server`, if you installed Indivo according to our instructions), run:

```
./migration_scripts/migrate_indivo.sh
```

And that’s it! Your database will now be ready for the newest version of Indivo.

Reset: If you don’t mind losing all existing data

Version 1.0 and above

- **Backup `utils/indivo_data.xml`**

  **Note:** If you are moving to version 2.X.X, apps are no longer defined in `indivo_data.xml`. Look at the latest `indivo_data.xml.default` for what can carry over, and then check out the new `registered_apps` directory for examples of how to migrate your apps in the new JSON format.

- **Copy `utils/indivo_data.xml.default` to `utils/indivo_data.xml`, and merge your old changes into that file**

  **Warning:** If you don’t move to the new data file, you might miss formatting changes that will prevent Indivo from loading your initial data.

- **Take a look at the options for the `indivo reset` script:**
  
  ```
  $ cd indivo_server
  $ python utils/reset.py --help
  ```

- **After a migration, it is usually helpful to drop and recreate the database with the `--force-drop` option. So try resetting the database, using:**
  
  ```
  $ python utils/reset.py --force-drop
  ```

If you have codingsystems data to load, add the `-c` flag:

```
$ python utils/reset.py --force-drop -c
```

Pre-Version 1.0

- **Backup `utils/indivo_data.xml`**
• Copy `utils/indivo_data.xml.default` to `utils/indivo_data.xml`, and merge your old changes into that file.

  **Warning:** If you don’t move to the new data file, you might miss formatting changes that will prevent Indivo from loading your initial data.

• Run the `indivo` reset script:
  
  ```
  $ cd indivo_server
  $ ./utils/reset.sh -sb
  ```

  If you have codingsystems data to load, add the `-c` flag:

  ```
  $ ./utils/reset.sh -sbc
  ```

### 6.5.4 Check the Release Notes

There may be version-specific issues to handle. Take a look at the [[Releases | Release Notes]]. Specifically, make sure to check that Indivo’s dependencies are all up to date. For example, in the 1.0.0 release, Indivo dropped support for Django version 1.1. Make sure to upgrade Django to version 1.2+ in order to run Indivo 1.0.0+

### 6.5.5 Run Some Sanity Checks

• Does the Indivo Test Suite run?:
  
  ```
  $ cd indivo_server
  $ python manage.py test indivo
  ```

• Do the development servers start up?:
  
  ```
  $ cd indivo_server
  $ python manage.py runserver 0.0.0.0:8000  # Or your preferred port to run Indivo Server on
  $ cd ../indivo_ui_server
  $ sudo python manage.py runserver 0.0.0.0:80  # Or your preferred port to run Indivo UI Server on
  ```

• Does Indivo seem to be working? Try logging in from a browser and clicking around.
7.1 Indivo X 1.0.0 Release Notes

The Indivo X 1.0.0 release is Indivo’s first production release, meant for production use. It has been tested extensively for security and stability, and is ready to support real data in the real world.

7.1.1 Grab the Code

- Indivo X Server
  * http://github.com/chb/indivo_server/
  * http://cloud.github.com/downloads/chb/indivo_server/indivo_server-v1.0.0.tar.gz
- Indivo X UI Server + bundled apps (problems, meds, labs)
  - http://github.com/chb/indivo_ui_server/
  - http://cloud.github.com/downloads/chb/indivo_ui_server/indivo_ui_server-v1.0.0.tar.gz
- Indivo X Python Client Library
  - http://github.com/chb/indivo_client_py/
  - http://cloud.github.com/downloads/chb/indivo_client_py/indivo_client_py-v1.0.0.tar.gz

7.1.2 Change Log

Indivo Server

Features

- New API calls to permanently delete app-specific documents and record-app-specific documents
- New API calls to rename and delete carenets
- Added Templates for activation, welcome, and forgot-password emails
- Performance fixes leading to an average 2x improvement in reporting calls.
- Large expansion of the Test Suite, including 80+ new tests, a new framework for loading test data, close to 9x faster test running time, and better management of testing transactional code.
- Better handling of database-level integrity errors
• Brand new documentation system written in Sphinx, with autogenerated API and source code documentation
• Reworked, neater settings file. MAKE SURE TO MOVE TO THE NEW FILE IF YOU ARE UPGRADING!!
• Added and tested compatibility with Django 1.3, python2.7, MySQL
• New reset script which avoids the need to drop/recreate the database every time
• New API calls to manage the autonomous app oAuth workflow

Behavior Changes

• Many calls that used to cause 500 errors now return 400s. Examples:
  – Attempting to reuse external_ids on documents, records, or messages
  – submitting invalid documents when XML validation is turned on
  – creating accounts with duplicate emails
• All XML returned from API calls is now stripped of whitespace between tags
• The 1-hour deletion rule for documents was removed: record-specific documents can NEVER be deleted.
• Document Status History now produces an iso8601 timestamp in the created_at field
• New Account email and username policy: all names are case insensitive.
• HTTP Methods are now enforced for all API calls. Making an API call with an unsupported HTTP method now results in an HTTP 405 error.
• The call to POST /records/{RECORD_ID}/shares/{OTHER_ACCOUNT_ID}/delete is deprecated. Use DELETE /records/{RECORD_ID}/shares/{OTHER_ACCOUNT_ID} (which is more RESTful) instead.
• The call to POST /records/{RECORD_ID}/notify is deprecated. Use POST /records/{RECORD_ID}/notifications/ (which is more RESTful) instead.
• The call to GET /id, which wasn’t ever very useful, has been removed.
• The XML describing Messages (returned from calls to the account inbox or to fetch individual messages) now describes attachments’ saved document ids with the ‘doc_id’ attribute, which is only present if the attachment has been successfully saved.
• Dropped support for Django 1.1. MAKE SURE TO UPGRADE to 1.2+ when you move to the new release.

Bugfixes

• Template Rendering of Reports now includes all Doc metadata
• Procedure XSLT now captures correct value for ‘location’
• Carenet filtering now works correctly with nevershares and implicit sharing
• Now enforcing unique message_ids per sender.
• DB errors due to failed unique constraints (i.e. duplicated external_ids) now return 400s not 500s
• Accepting Message attachment returns 400 on documents that don’t validate, not 500
• API calls to invalid URLS now return 405 instead of a misleading 403
• Getting latest documents now sorts in the correct order
• Document metadata correctly renders the id of the document it replaces.
• Document Status History now produces an iso8601 timestamp in the created_at field
• Now attempting to version a document that isn’t the latest version raises an error
• Document creation by relationship now returns a 400 if the new doc fails validation, not a 404
• Paging of results is no longer nondeterministic when breaking ties on the primary ordering
• Attempting to assign the same name to two different carenets now returns a 400, not a 500
• Creating accounts with duplicate emails returns a 400, not 500
• Document Size is now calculated properly, and not overwritten when the doc is processed.
• Document Types are now assigned to all XML docs, even ones that don’t match the core schemas
• Replacing Documents now updates their MIME types correctly
• Resolved inefficient DB access in reporting calls
• When XML Syntax Validation is on, now validates all documents, not just those that match core schemas
• HTTP Methods are now enforced for all API calls
• XML describing Indivo Messages now reports attachment size correctly, and renders dates in iso8601 format.
• XML describing Notifications now outputs dates in iso8601 format
• Tests no longer fail if settings.MEDIA_ROOT hasn’t yet been created in the filesystem
• Get-latest-document algorithm no longer relies on timestamps (which might be tied)
• Creating binary files no longer saves their content twice

Indivo UI Server

• New configuration options for:
  – allowing users to add records
  – controlling the ability of users to register, and the level of involvement required by administrators
  – minimum password length
  – automatically creating a first record for accounts
• A New Sharing Management interface, with:
  – drag and drop functionality
  – The ability to create, delete, and rename carenets
  – The ability to add and remove accounts from carenets
• A New App Settings interface, with:
  – Drag and drop functionality
  – The ability to enable and disable apps
  – The ability to add or remove apps from carenets
• Updated JavaScriptMVC components to version 3.2
• Increased error handling and message/status displays
• Improvements to Inbox usability and functionality
• Improvements to workflow for non-framed apps
Python Client

Features

• New custom call to create records, with or without pre-built contact XML
• Added call to verify_surl()
• Added calls to get reports from within a carenet
• Added calls to get info about a userapp and list all userapps
• Added call to enable a userapp on a record
• Added calls for the new autonomous app oauth workflow
• Added call to get the owner of a record
• Added a call to unshare a userapp from a carenet

Behavior Changes

• create_session() now raises exceptions if the response status wasn’t a 200
• removed unimplemented/deprecated calls to /oauth/authorize and /oauth/access_token
• delete_share and record_notify no longer user deprecated paths

Bugfixes

• create_account() now uses all POST data, including contact emails
• account_secret_resend now uses correct HTTP method
• account_forgot_password() now uses correct path

7.1.3 Some Things You Should Know

• This release Drops support for Django 1.1. MAKE SURE TO UPGRADE TO 1.2+ when you move to the new release.
• Because of licensing restrictions, we are not able to distribute SNOMED and UMLS concept codes, so the codingsystems functionality is not very developed right now.

7.2 Indivo X 2.0.0 Release Notes

The Indivo X 2.0.0 release represents a major step forward in interoperability by providing a deep integration of the SMART data models and API. SMART apps will now be able to run unaltered on Indivo, or enhanced to take advantage of Indivo’s full API. Also included in this release is a new system for easily extending Indivo with custom data models, and the ability to retrieve them in multiple formats (XML, JSON, RDF/XML).
7.2.1 Grab the Code

- Indivo X Server
  - http://github.com/chb/indivo_server/
  - http://cloud.github.com/downloads/chb/indivo_server/indivo_server-v2.0.0.tar.gz
- Indivo X UI Server + bundled apps (allergies, labs, medications, problems)
  - http://github.com/chb/indivo_ui_server/
  - http://cloud.github.com/downloads/chb/indivo_ui_server/indivo_ui_server-v2.0.0.tar.gz
- Indivo X Python Client Library
  - http://github.com/chb/indivo_client_py/
  - http://cloud.github.com/downloads/chb/indivo_client_py/indivo_client_py-v2.0.0.tar.gz

7.2.2 Change Log

Indivo Server

- Now supports the SMART API
- New methods for easily defining new data models and attaching custom serializers and validators
  - Django Model Classes
    * For those with python/Django experience
  - Simple Data Modeling Language (SDML)
    * JSON-based modeling language for defining simple data models
- Reports over all data models, supporting the API Query Interface, and with XML and JSON serialization options
- New SMART-style JSON app manifests
- Sample data profiles for demo and testing purposes

Indivo UI Server

- Now supports SMART apps
- New Labs app

Python Client

- New client using the python-oauth2 library
- Full access to all response information

7.2.3 Some Things You Should Know

- With our adoption of the SMART data models and API, we have removed overlapping Indivo models. This means that there is NO MIGRATION PATH from the 1.x.x to the 2.x.x version of Indivo.
Indices and tables

- genindex
- modindex
- search
## API Call Lookup Table

### /accounts

- GET /accounts/{0}/inbox/{1}, ??
- GET /accounts/{0}/notifications/,
- GET /accounts/{0}/permissions/,
- GET /accounts/{0}/primary-secret, ??
- GET /accounts/{0}/check-secrets/{1}, ??
- GET /accounts/{0}/apps/{1}/preferences, ??
- GET /accounts/{0}/secret, ??
- GET /accounts/search, ??
- GET /accounts/{0}/inbox/, ??
- GET /accounts/, ??
- PUT /accounts/{0}/apps/{1}/preferences, ??
- POST /accounts/{0}/authsystems/password/set, ??
- POST /accounts/{0}/authsystems/password/change, ??
- POST /accounts/{0}/authsystems/password/set-username, ??
- POST /accounts/{0}/initialize/{1}, ??
- POST /accounts/{0}/reset, ??
- POST /accounts/, ??
- POST /accounts/{0}/forgot-password, ??
- POST /accounts/{0}/set-state, ??
- POST /accounts/{0}/info-set, ??
- POST /accounts/{0}/inbox/{1}/archive, ??
- POST /accounts/{0}/inbox/{1}/attachments/{2}/accept, ??
- POST /accounts/{0}/inbox/, ??
- DELETE /accounts/{0}/apps/{1}/preferences, ??

### /apps

- GET /apps/manifests/, ??
- GET /apps/,
- GET /accounts/{0}/records/, ??
- GET /accounts/{0}/documents/external/{1}/meta, ??
- GET /accounts/{0}, ??
- GET /accounts/{0}/manifest, ??
- GET /accounts/{0}/documents/{1}, ??
- GET /accounts/{0}/documents/, ??
- PUT /accounts/{0}/documents/external/{1}, ??
- PUT /accounts/{0}/documents/{1}, ??
- PUT /accounts/{0}/documents/{1}/label, ??
- POST /accounts/{0}/records/{1}/access_token, ??
- POST /apps/{0}/documents/, ??
- DELETE /apps/{0}, ??
- DELETE /apps/{0}/documents/{1}, ??

### /capabilities

- GET /capabilities/, ??

### /carenets

- GET /carenets/0/reports/, ??
- GET /carenets/0/accounts/1/permissions, ??
- GET /carenets/0/apps/1/permissions, ??
- GET /carenets/0/reports/minimal/procedures/, ??
- GET /carenets/0/documents/??
- GET /carenets/0/reports/minimal/simple-clinical-notes/??
- GET /carenets/0/documents/external/??
- GET /carenets/0/apps/??
- GET /carenets/0/documents/??
GET /carenets/{0}/accounts/,
GET /carenets/{0}/report/,
GET /carenets/{0}/documents/{1}/meta,
GET /carenets/{0}/reports/minimal/measurement/{1}/,
PUT /carenets/{0}/apps/{1}/,
POST /carenets/{0}/rename/,
DELETE /carenets/{0}/accounts/,
DELETE /carenets/{0}/,
DELETE /carenets/{0}/apps/1/,
/codes
GET /codes/systems/{0}/query/,
GET /codes/systems/,
/oauth
GET /oauth/internal/surl-verify/,
GET /oauth/internal/request_tokens/{0}/info/,
POST /oauth/internal/request_tokens/{0}/approve/,
POST /oauth/request_token/,
POST /oauth/access_token/,
POST /oauth/internal/session_create/,
/ontology
GET /ontology/,
/records
GET /records/{0}/reports/{1}/,
GET /records/{0}/search/,
GET /records/{0}/allergies/,
GET /records/{0}/documents/{1}/meta/,
PUT /records/{0}/documents/external/{1}/{2}/label/,
PUT /records/external/{0}/{1}/,
PUT /records/{0}/owner/,
PUT /records/{0}/documents/1/rels/2/3/label/,
PUT /records/0/documents/1/nevershare/,
PUT /records/0/documents/external/1/2/label/,
PUT /records/{0}/apps/{1}/documents/{2}/label,
  ??
PUT /records/{0}/documents/{1}/carenets/{2},
  ??
POST /records/{0}/documents/{1}/set-status,
  ??
POST /records/{0}/inbox/{1}/attachments/{2},
  ??
POST /records/{0}/apps/{1}/documents/external/{2},
  ??
POST /records/{0}/documents/{1}/rels/{2}/external/{3}/{4},
  ??
POST /records/{0}/documents/{1}/replace,
  ??
POST /records/{0}/documents/{1}/rels/{2}/,
  ??
POST /records/{0}/notify,
  ??
POST /records/{0}/autoshare/carenets/{1}/bytype/set,
  ??
POST /records/{0}/notifications/.,
  ??
POST /records/{0}/autoshare/carenets/{1}/bytype/unset,
  ??
POST /records/{0}/apps/{1}/documents/,
  ??
POST /records/{0}/shares/.,
  ??
POST /records/.,
  ??
POST /records/{0}/shares/{1}/delete,
  ??
POST /records/{0}/apps/{1}/setup,
  ??
POST /records/{0}/documents/.,
  ??
DELETE /records/{0}/documents/{1}/nevershare,
  ??
DELETE /records/{0}/apps/{1},
  ??
DELETE /records/{0}/apps/{1}/documents/2,
  ??
DELETE /records/{0}/shares/{1},
  ??
DELETE /records/{0}/documents/{1}/carenets/{2},
  ??
DELETE /records/{0}/documents/,
  ??

/version
GET /version,
a
indivo.views.account,??
indivo.views.audit,??
views.account,??
views.audit,??

b
indivo.views.base,??
views.base,??

d
indivo.views.documents,??
indivo.views.documents.document,??
indivo.views.documents.document_delete,??
indivo.views.documents.document_label,??
indivo.views.documents.document_meta,??
indivo.views.documents.document_rels,??
indivo.views.documents.document_status,??
indivo.views.documents.document_versions,??
indivo.views.documents.special_documents,??
views.document.document,??
views.document.document_delete,??
views.document.document_label,??
views.document.document_meta,??
views.document.document_rels,??
views.document.document_status,??
views.document.document_versions,??

i
indivo,??
indivo.data_models,??
indivo.data_models.contrib,??
indivo.data_models.core,??
indivo.data_models.options,??
indivo.fields,??
indivo.fields.dummy_fields,??
indivo.serializers,??
indivo.serializers.base,??
indivo.serializers.datamodel_hooks,??
indivo.serializers.json,??
indivo.serializers.python,??
indivo.serializers.xml_serializer,??
indivo.validators,??
indivo.views,??
indivo_python_client,??

m
indivo.views.messaging,??
views.messaging,??

p
indivo.views.pha,??
views.pha,??

r
indivo.views.record,??
indivo.views.reports,??
indivo.views.reports.ccr,??
indivo.views.reports.equipment,??
indivo.views.reports.generic,??
indivo.views.reports.measurement,??
indivo.views.reports.procedure,??
indivo.views.reports.simple_clinical_notes,??
indivo.views.reports.smart,??
views.reports.ccr,??
views.reports.equipment,??
views.reports.generic,??
views.reports.measurement,??
views.reports.procedure,??
views.reports.simple_clinical_notes,??
views.reports.smart,??
S
indivo.views.shares,??
indivo.views.shares.shares_account,??
indivo.views.shares.shares_autoshare,??
indivo.views.shares.shares_carenet,??
indivo.views.shares.shares_document,??
indivo.views.shares.shares_nevershare,??
indivo.views.shares.shares_pha,??
indivo.views.smart_container,??
views.sharing.shares_account,??
views.sharing.shares_autoshare,??
views.sharing.shares_carenet,??
views.sharing.shares_document,??
views.sharing.shares_nevershare,??
views.sharing.shares_pha,??

V
view.smart_container,??