[Introduction](#h.u2udtf7p0wac)

[ISF SVN organization](#h.ovlmp6ucc8xx)

[ISF modules description](#h.v57u90vqwvio)

[ISF Vocabulary model](#h.kt81u8xse1xd)

[Reified relationship model](#h.u68fonqfj436)

[ERO and VIVO migration/refactoring](#h.62je6xvrmqsi)

[Entity and axiom migration](#h.78zci8nw329u)

[Ontology vs. application separation](#h.43lguj4xnhbc)

[Ontology Browser](#h.c8xoofqzgwfj)

[Future work](#h.sf0ferxzbodx)

## Introduction

In this document we present a description of the content, the design, and modeling choices behind the beta version of the ISF framework released on 5/31/2013.

## ISF SVN organization

The ISF beta release is composed of a set of files hosted at the “connect-isf” Google Code project [1]. This project uses Subversion (SVN) for its code repository. The following describes the current content of the files and layout in the SVN repository.

Figure 1 shows the SVN directory layout. The top directory is “ontology” and it contains the “arg.owl” OWL file that imports all necessary files (from various subdirectories) to give a full view of the ISF. This directory has the following four subdirectories:

* The “module” folder contains files that cover the main content of the ISF ontology and is described in the next section.
* The “app-views” directory has application-specific files that import one or more of the application-independent ISF files from the other directories (imports, mireot, module). The “eagle-i” subdirectory contains files specific to the eagle-i application [2] and the “vivo” subdirectory is for the VIVO application [3].
* The “imports” directory contains copies of external ontologies (such as VCARD or Basic Formal Ontology) that are used (through owl:imports directives) by one or more of the other files. This directory is intended to serve as a cache for OWL files that are used in OWL imports so that they can be resolved locally when needed. This also means that this directory could contain an ontology document for a specific ontology but the document returned on the web might be different if it has been recently updated.
* The “mireot” directory is for MIREOTed [4] files. A MIREOT file is a file that contains some OWL content (classes, properties, annotations, etc.) coming from other existing ontologies (such Ontology for Biomedical Investigations, OBI or the Gene Ontology, GO) that are used in the ISF ontology - but not the whole ontology as in the case of the imports directory. This approach helps avoid having to import full ontologies and the scripted nature of this approach also helps with the maintenance of these files.

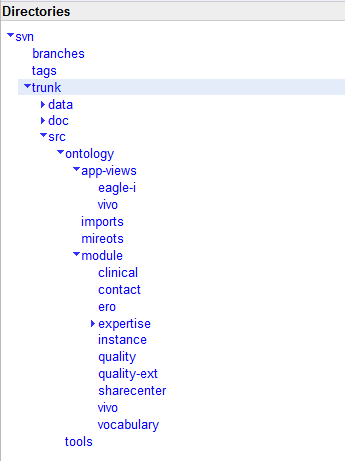
Other SVN directories organize ISF related documentation (doc folder), tools, and data.

The tools folder will contain custom scripts that will be developed for the final release of the ontology while the data folder is meant for any related data files such as ISF-compliant RDF data or examples to demonstrate the correct use of the ISF.

The SVN layout makes a distinction between ontology content that should be reusable and content that is application specific. The final release of the ISF will include some reorganization of these folders with an additional “source” subdirectory that will provide an additional level of organization in addition to the two already described (ontology vs. application). This will be detailed in the final documentation.

The main file that collects all the other files through a owl:import chain is the arg.owl under the ontology folder (this file will be renamed isf.owl in the final release).

All the classes, properties, and individuals in the various files that have an identifier in the form of ARG\_XXXXXXX are the entities we have created during the CTSAconnect project to meet some requirements that couldn’t be met by reusing entities already existing in other ontologies. The prefix ARG stands for Agent, Research resources and Grants, and is a unique prefix for use in the OBO Foundry [5].



**Figure 1 . The tree structure of ISF SVN repository.**

## ISF modules description

The scope and size of the ISF ontology is large and is expected to grow over time. Therefore, we are developing a modular approach that should simplify the reuse of the ISF ontology by end-users. The general principle is that the released version of ISF will provide stand alone modules for specific use-cases.

An analogy would be how relational databases organize data into distinct tables (but with inter-table references where needed, i.e. foreign key references) but also provide use-case specific views over the same data. Also, these views usually contain redundant or denormalized data derived from the data in the tables. Our long-term goal is to follow this organizing principle where certain OWL files and directories represent the native or source data in a normalized non-overlapping fashion while still being able to provide redundant and possibly overlapping modules for specific use-cases.

This approach is still in early development and it will be more complete for the first official release of the ISF. Table 1 describes the current status of this modular approach, which is represented in SVN as subdirectories of the “module” directory. We will likely reorganize the files under the “module” folder during the final release to fully remove any original VIVO and ERO files and group content according to the final released modules coverage.

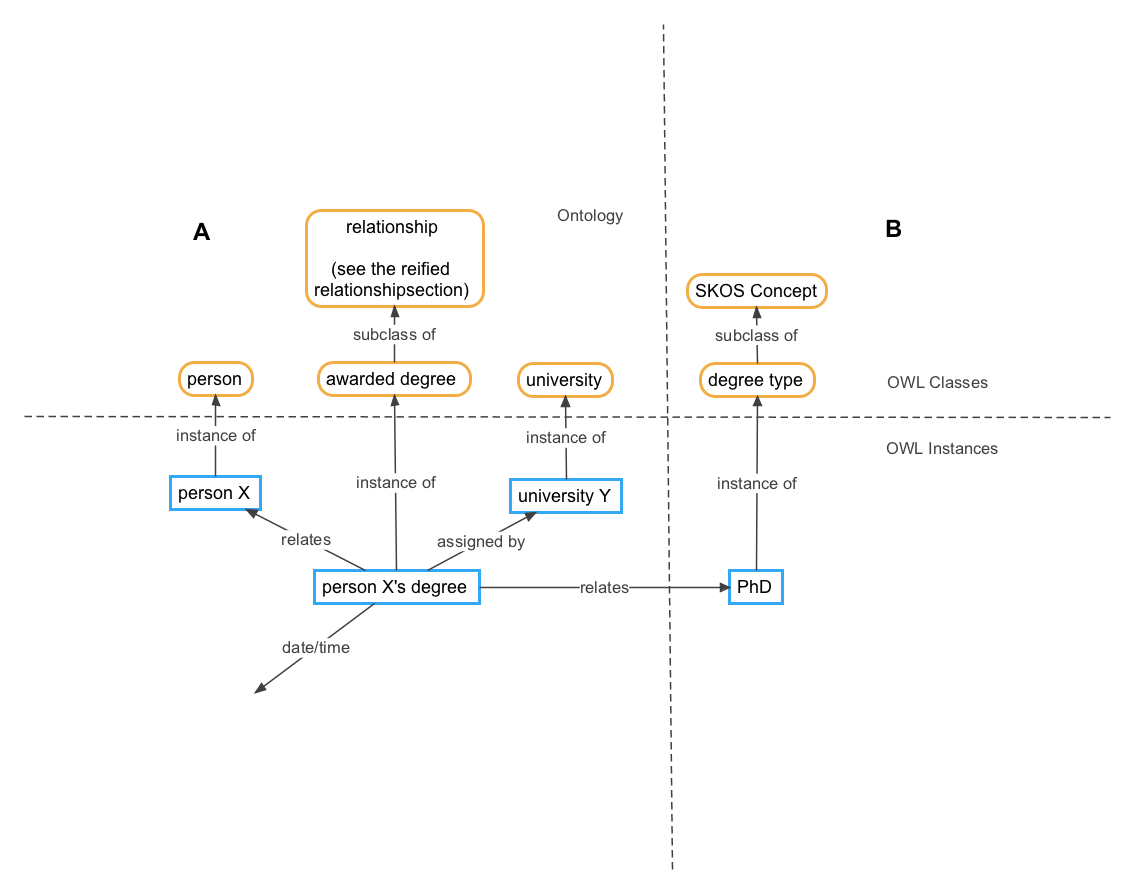
|  |  |
| --- | --- |
| **Module folder name** | **Description** |
| clinical | This module contains entities that are used to represent clinical encounters |
| contact | This module contains the entities that describes agents (people and organizations) contact information |
| ero | In this module we collect research resources covered by the eagle-i ontology that have been reused in ISF |
| expertise | This modules defines the entities used to represent clinical expertise based on clinical encounter data |
| instance | This module contains OWL instances for specific medical boards |
| quality/ quality-ext | This module currently holds qualities, such as experience, that will be moved from current VIVO and ERO files during the final release |
| sharecenter | This module contains OWL files for classes that link Sharecenter tags and RDF data to the ISF |
| vivo | In this module we collect resources covered by the VIVO ontology that have been reused in ISF |
| vocabulary | This module contains sets of terminologies used by other modules such as the ICD-9 codes used by the expertise module. See the vocabulary section below. |

**Table 1. ISF beta modules (represented at sub-directories under the “module” directory)**

## ISF Vocabulary model

An ontology is mostly composed of classes of things (types, sets, etc.) from the domain of interest. These classes are then instantiated by creating class members (individual things) to represent the details of the domain. These individuals are given unique identifiers and are related to each other accordingly. However, there is also a need for being able to reference a concept in a way that is different from referencing individual instances. The following example clarifies this issue and describes our approach. This was one of the major goals discussed in the proposal - to develop a standardized mechanism to leverage non-ontological vocabularies.

In the ISF there is a class labeled “awarded degree” and instances of this class are created when a degree is created or given to a person. The individual degree instances are uniquely identified for every degree given to a person. This class, in ISF, is meant to relate a unique degree to the organization issuing it, the person receiving it, time values, location, etc. (it is a representation of an n-ary relation) but instances of this class need to also specify the type of degree received. One option is to create additional classes in the ontology for every possible degree type (either as subtypes of “awarded degree” or as a separate branch) and instantiate the type for each instance of an “awarded degree”. This level of granularity was not found to be useful in our ISF modeling and our approach was to create a “degree type” class and create a single shared instance for each type (vs. creating classes for each type). We consider these instances to be a vocabulary for the various degree types and this vocabulary can be used wherever there is a need to refer to a “degree type”. An instance of “awarded degree” is then related to one of the vocabulary entries to specify the type of awarded degree. The diagram in Figure 2 shows this approach.



**Figure 2. Use of vocabularies in the context of the ISF, illustrated using the awarded degree example.** An instance of the “awarded degree” class that belong to the “person X”, which is an instance of the “person” class. This awarded degree is also related to a specific university instance and to the vocabulary instance “PhD” that is an instance of the SKOS concept “degree type” (see the SKOS and “reified relationship” explanations below).

The SKOS ontology is an ideal option for representing this type of knowledge (where concepts are represented as individuals/instances instead of classes) and we use it in the ISF to organize our vocabularies. The top SKOS class “Concept” is subclassed for a specific vocabulary and then class instances are created for each vocabulary entry. We are still developing this model, but we are currently using it for ICD9, MeSH, and UMLS codes (with appropriate SKOS relationships) and few other vocabularies being considered for the final release.

## Reified relationship model

The ISF ontology is based on the Basic Formal Ontology (BFO) upper ontology [6]. BFO provides a model for how independent continuant entities (objects, people, and other physical things) can interact with each other over time in processes. A process is, more or less, a dynamic interaction or relationship between things and it usually implies actual activities and change over some period of time. The reified relationship model is our approach for representing relationships that are not processes (and that do not imply ongoing activities or change) but still exist over some time.

For example, if a student is assigned a mentor in an educational program, we would like to be able to capture and represent this relationship in our data. The RDF data model provides binary relationships and one can represent this example with a RDF statement such as “student\_x *has\_mentor* mentor\_y” where the “*has\_mentor”* is a directed binary relationship (a line) between the two people (the nodes) involved in the relationship. However, this binary relationship does not allow us to also capture the date the relationship started or ended, who assigned the mentoring relationship, and any other related facts. However, if we represent the “*has\_mentor”* relationship as a first-class instance (a node instead of a line), we will be able to relate other things to the relationship. This also allows the relationship instance to have a unique identifier so that it can be referenced in other data if needed. This is the approach we take in the ISF and similar use is found in other projectssuch as Ontologyx [7].

The above allows us to represent a “mentorship” relationship without implying that there are any “mentoring processes”. When actual mentoring occurs, this can be captured with BFO processes, roles, and people. However, for the static aspect of the relationship we need this “reified relationship” approach. Figure 2 shows this for the “awarded degree” relationship. The instance “person X’s degree” is an instance of a reified relationship and it relates a person, a university, time/date, and the shared “PhD” degree type vocabulary entry.

This model provides two general binary relationships to relate a reified relationship to other things. There is the “assigns” and “assigned by” pair that relates a reified relationship to an agent who asserted, created, etc. the relationship. The other general binary relationship is the “relates” and “related by” that captures the things that are being related in the reified relationship. The general model is very simple, it allows us to capture statements such as “agent\_x created a relationship between things” but it is very flexible and allows for composing several relationships with each other to represent very complex data. Figure 2 above shows these relationships to indicate that a specific university was the agent that created the degree instance for a specific person and a specific degree type.

## ERO and VIVO migration/refactoring

As described in the introduction, the main content of ISF is based on merging the ERO and VIVO ontologies. The following describes how this merge is implemented in the SVN repository.

A copy of the original ontology files of VIVO and ERO was obtained and placed in the “vivo” and “ero” modules folders respectively. ERO also had a separate set of “application” OWL files that contained additional content relevant for the use of the ERO ontology in the eagle-i application. These files were placed in the “eagle-i” directory under the “app-views” directory.

A top level OWL file was created for each of ERO and VIVO (named ero.owl and vivo.owl) in the corresponding modules. These files import the parts of ERO and VIVO ontologies that are being reused by ISF. Application specific files named “ero-app.owl” and “vivo-app.owl” were also created in the corresponding “app-views” directory and they import the corresponding ontology files “ero.owl” and “vivo.owl”. These application specific files contain mostly sets of annotations that allow correct displaying of ISF content under eagle-i and VIVO applications.

A top importer file named arg.owl (to be renamed isf.owl) in the “ontology” directory imports the above two files (not the application specific files) and the top OWL for each module. This file gives a full view over the ISF ontology content. A corresponding “arg-app.owl” is created and placed in the “eagle-i” application directory for the purpose of being able to browse the ISF ontology in the eagle-i ontology browser.

### Entity and axiom migration

After the initial merge described above, the ISF content is being reviewed and refactored as needed. The result of this ongoing process is that a set of OWL entities (classes, properties, etc.) and OWL axioms (logical definitions or annotations) are being removed from ISF. This occurs when there is an overlap between ERO and VIVO, or when one or the other, or both, are migrating their logical representation to a new representation developed during the refactoring process. The new representations are meant to be more detailed and generalized so that they can be applied to a wider range of use-cases.

When an entity or axiom is removed, it might still be needed by an ERO or VIVO based application until the application is also migrated. To support this possibility, the entities and axioms are pushed to the corresponding ero-app.owl and vivo-app.owl files so that an application (currently eagle-i and VIVO applications) can still have an application view that does not appear to be changing. The process of pushing an axiom from ISF to an application specific file is called “ISF deprecation”. An “ISF deprecated” entity or axiom could still be an extension of an ISF entity, but the extension is considered too specific and, for the time being, is being pushed to an application view until wider need for the entity or axiom becomes apparent. At that point, the entity or axiom can be moved back to ISF. Other entities and axioms are truly “ISF deprecated” and will not be incorporated in ISF until the specific deprecation reason is resolved. For example, an OWL class could have been vague and used with multiple distinct meanings. Such a class needs to be re-represented with multiple classes (one of which could be the original class) with clear and distinct definitions. At that point, one or more of the re-represented classes could be incorporated into ISF.

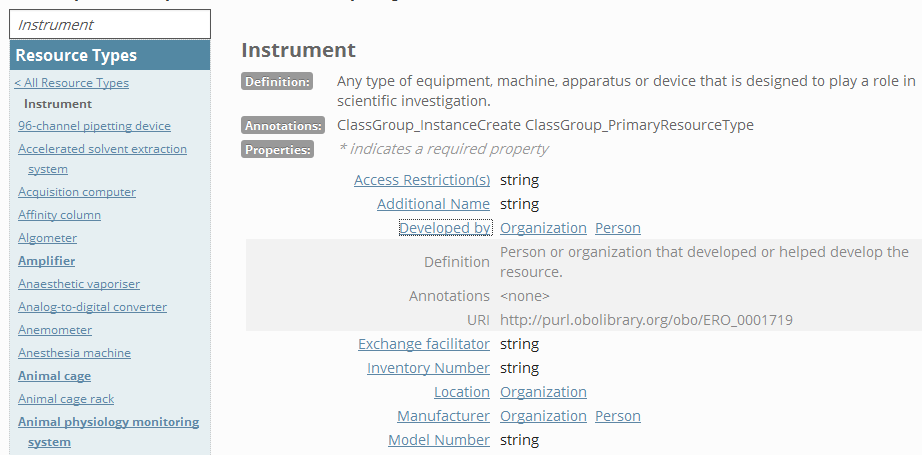
A subset of the “ISF deprecated” entities and axioms will be fully deprecated and not used by any applications or data. At that point, they will be moved to the ero-deprecated.owl and vivo-deprecated.owl files in the same directories where the application files exist.

## Ontology vs. application separation

Any application that uses the ISF ontology will likely need additional OWL content to support the specific application. This content can be extensions of ISF classes, preferred annotations, equivalence axioms, etc. The current SVN repository includes this content for the eagle-i and VIVO applications under the “app-views” directory. Other applications such as Plumage [8] would be welcome to include such application configurations in this directory.

## Ontology Browser

The eagle-i ontology browser is being used to visualize the ISF ontology. It can be accessed at a development instance at <http://ohsu.dev.eagle-i.net/model/>. The following describes this UI and how it relates to the OWL content of the ISF ontology. Figure 3 shows this UI with the “Instrument” class selected.

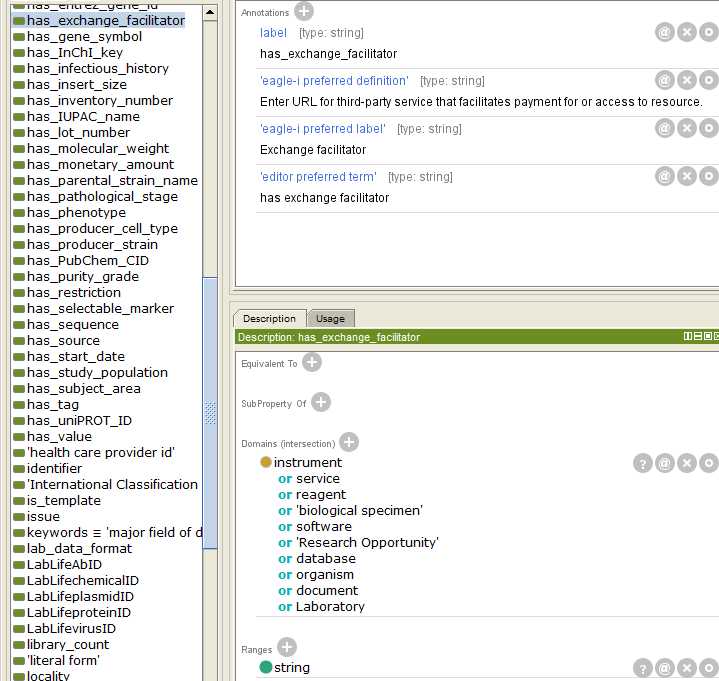


**Figure 3. The ISF visualized through the eagle-i ontology browser.**

The search box above the left navigation bar can be used to find classes from the ISF ontology. The “Resource Types” section of the navigation bar is used to navigate the class hierarchy. Instrument is selected and any classes above it are parent classes and the different instrument subclasses are shown below it. The main content shows the selected class, its text definition, and any relevant properties. A property can be clicked to open an area that shows additional information about the property. In the above screenshot we see that an instrument can have the listed properties and we also see the possible values of the properties.

The applicable properties for a specific class are calculated based on the logical definitions in the ontology and additional application-specific annotations. The annotations are used when the ontology itself does not provide this level of detail. An ontology usually does not define the various attributes needed to collect information about an instance of a class (a specific instrument in a specific lab) but it does provide the classes and logical relationships needed to capture this information. The above image shows that the eagle-i application chose to collect information about an “exchange facilitator” for instrument but this same property might be also used to capture similar information about other classes.

It is important to keep the above notes in mind when navigating the ontology browser and comparing it with the ontology content in an editor such as Protégé. In Protégé, when opening the arg-app.owl file in the beta release folder [9], the class “Instrument” has a label of “instrument” and an eagle-i preferred label (in an application file) of the capitalized form “Instrument”. Also, the properties shown above do not appear as logical definitions of the instrument class. Instead, individual properties in the ontology indicate (either through logical definitions or application specific annotations) which classes they relate to. For example, the Protégé screenshot in Figure 4 shows that the “Exchange facilitator” property applies to instruments and few other classes and its value should be a string. It also shows few eagle-i application annotations in addition to the label annotation, including a user-interface “eagle-i preferred label” and definition.



**Figure 4 . A Protégé screenshot showing the application-specific labels for the *has\_exchange\_facilitator* property.**

## Future work

For the final release of ISF, we will be reorganizing the current files under the modules folder in a way that will remove eagle-i and vivo files and group the current classes and properties and related axioms in a set of files that covers particular content.

This new file organization will be leveraged to produce a set of “released” modules that could be self contained and could be reused by other ontologies and applications. We will be developing scripts to automate the file reorganization and the generation of the modules.

[1] <https://code.google.com/p/connect-isf/>

[2] <https://www.eagle-i.net/>

[3] <http://vivoweb.com/>

[4] Courtot M, Gibson F, Lister A, et al: MIREOT: the Minimum Information to Reference an External Ontology Term. In Proc. ICBO’09 2009. <http://icbo.buffalo.edu/2009/Proceedings.pdf#page=101>

[5] <http://www.obofoundry.org/>

[6] <https://code.google.com/p/bfo/>

[7] G. Rust. Ontologyx. In Functional Requirements for Bibliographic Records Workshop Proceedings, Dublin, Ohio, May 2005.

[8] <https://github.com/CTSIatUCSF/plumage>

[9]<https://connect-isf.googlecode.com/svn/tags/1.0.0_beta/src/ontology/app-views/eagle-i/arg-app.owl>