IHTSDO Workbench

Keith Campbell
Susan Castillo, Eric Browne
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting Preferences</td>
<td>95</td>
</tr>
<tr>
<td>The Preferences Tabs</td>
<td>96</td>
</tr>
<tr>
<td>View Tab</td>
<td>96</td>
</tr>
<tr>
<td>Descriptions</td>
<td>96</td>
</tr>
<tr>
<td>Hierarchy Roots</td>
<td>96</td>
</tr>
<tr>
<td>Taxonomy</td>
<td>97</td>
</tr>
<tr>
<td>Taxonomy sort</td>
<td>97</td>
</tr>
<tr>
<td>Status</td>
<td>97</td>
</tr>
<tr>
<td>Refsets to show in taxonomy view</td>
<td>97</td>
</tr>
<tr>
<td>Edit Tab</td>
<td>97</td>
</tr>
<tr>
<td>Defaults</td>
<td>97</td>
</tr>
<tr>
<td>Relationship Types (values for popup)</td>
<td>98</td>
</tr>
<tr>
<td>Relationship Refinability (values for popup)</td>
<td>98</td>
</tr>
<tr>
<td>Relationship Characteristics</td>
<td>98</td>
</tr>
<tr>
<td>Description Types</td>
<td>98</td>
</tr>
<tr>
<td>Image Types</td>
<td>98</td>
</tr>
<tr>
<td>Status Values</td>
<td>98</td>
</tr>
<tr>
<td>Path Tab</td>
<td>98</td>
</tr>
<tr>
<td>New Path Tab</td>
<td>98</td>
</tr>
<tr>
<td>Reference Set Tab</td>
<td>98</td>
</tr>
<tr>
<td>Component Panel Tab</td>
<td>99</td>
</tr>
<tr>
<td>Adding and Modifying Concepts</td>
<td>101</td>
</tr>
<tr>
<td>Concepts</td>
<td>102</td>
</tr>
<tr>
<td>Relationships</td>
<td>105</td>
</tr>
<tr>
<td>Submitting a Change Set to the IHTSDO Workspace</td>
<td>107</td>
</tr>
<tr>
<td>Data Checks and Search Filters</td>
<td>109</td>
</tr>
<tr>
<td>Searching for a Concept</td>
<td>110</td>
</tr>
<tr>
<td>DataChecks</td>
<td>111</td>
</tr>
<tr>
<td>Reference Sets</td>
<td>113</td>
</tr>
<tr>
<td>Creating a Member Refset Manually</td>
<td>114</td>
</tr>
<tr>
<td>Creating a Specification Refset Manually</td>
<td>114</td>
</tr>
<tr>
<td>Associating Concepts with a Specification Reference Set</td>
<td>115</td>
</tr>
<tr>
<td>Displaying Specification Refset Concept Types in the Hierarchy List</td>
<td>116</td>
</tr>
<tr>
<td>Displaying Refset Members in the Hierarchy List</td>
<td>117</td>
</tr>
<tr>
<td>Changing the Refset Default Values</td>
<td>117</td>
</tr>
</tbody>
</table>
# The Terminology Validator Role

The Terminology Validator Role is responsible for validating terminologies to ensure they meet the required standards and specifications. This role involves checking for consistency, completeness, and adherence to standards such as SNOMED CT. The validator must also ensure that the terminologies are updated regularly to reflect the latest medical knowledge and terminology changes.

# Setting up Eclipse for IHTSDO Workbench Development

Setting up Eclipse for IHTSDO Workbench Development requires the installation of necessary plugins and configurations. This involves downloading the latest version of Eclipse from the official website, configuring the workspace, and installing the IHTSDO Workbench plugins. Additionally, ensure that the necessary permissions and configurations are set up correctly to facilitate smooth development.

# About the IHTSDO Workspace

About the IHTSDO Workspace provides an overview of the workspace's components and features. It covers aspects such as integrated and on-demand services, configuration management using CollabNet Subversion, layered API library, application lifecycle management using CollabNet SourceForge, and build, test, and publish services management using CollabNet CUBiT. Additionally, it includes smart-client integration.

# Creating new tasks

Creating new tasks involves planning, execution, and documentation. Example Task Extensions provide guidelines for extending the basic functionality of tasks. Steps to Create A Task outline the process of creating a new task, including defining the task's purpose, setting objectives, and determining the necessary resources.

# Building an application bundle

Building an application bundle involves creating a distributable package that can be deployed on various platforms. Bundle structure refers to the organization of files and directories necessary for the bundle to function correctly. Setting up for Maven builds involves configuring Maven to compile and package the application bundle according to the specified requirements. Distribution and update via Continuum allows for seamless deployment and updates of the bundle.

# Maven Plug-ins

Maven Plug-ins provide extensions to the standard Maven functionality, allowing developers to perform tasks such as packaging, testing, and deploying the application bundle. DWFA Maven Plug-ins are specifically designed for the Development Workspace for Terminology Application (DWFA) and provide additional features for managing and deploying terminologies.

# Using the IHTSDO Workspace on Collabnet

Using the IHTSDO Workspace on Collabnet involves accessing and utilizing the workspace on the Collabnet platform. It includes using the IHTSDO Workspace on Collabnet, which provides a collaborative environment for development and testing.

# Data Import

Data Import involves importing terminology sets into a versioned database. Terminology Set imports to Versioned Database outline the process of importing terminology sets, ensuring compatibility and integration with the workspace.

# Toolkit Support and Trouble-shooting

Toolkit Support and Trouble-shooting provides assistance and solutions to common issues encountered while using the IHTSDO Workbench Toolkit. How to determine the SNOMED code of a concept or term?, How to determine the version of a term or a terminology being edited?, How to determine the version of the IHTSDO Workbench Toolkit being used?, How to chain together tasks in a business process/workflow?, How to extend the IHTSDO Workbench Editor menu?, How to set user profiles?, How to overcome memory issues?, How to recover from a corrupt profile?
An IHTSDO Collaborative Environment for developing quality terminologies.
February 2009

Acknowledgements:

This Guide, and the IHTSDO Toolkit it describes are the result of collaborative contributions from a number of organisations. The publishers hereby acknowledge and thank the following organisations:

Informatics Inc., Bend, Oregon USA.
http://www.informatics.com

National Centre for Classification in Health, Sydney Australia.
http://www.ncch.com.au

National E-Health Transition Authority, Sydney, Australia

Australian e-Health Research Centre, Brisbane, Australia
http://e-hrc.net

Editors:
Keith Campbell M.D. Ph.D.
Susan Castillo
Eric Browne

Publisher:
Copyright IHTSDO 2009
International Health Terminology Standards Development Organisation
Rued Langgaards Vej 7, 5, 2300 Copenhagen S, Denmark
http://www.ihtsdo.org
IHTSDO Workbench Guide: Introduction and Overview

Purpose, Scope, Boundaries and Requirements

<table>
<thead>
<tr>
<th>Title:</th>
<th>IHTSDO Workbench Guide: Introduction and Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date:</td>
<td>20090420</td>
</tr>
<tr>
<td>Version:</td>
<td>0.04</td>
</tr>
<tr>
<td>Creating Authors:</td>
<td>Keith Campbell, Susan Castillo, Eric Browne</td>
</tr>
<tr>
<td>Subject:</td>
<td>IHTSDO, Workbench, introduction</td>
</tr>
</tbody>
</table>

Subject should be filled in as three keywords. The first keyword should be a structural or organizational entity, e.g. IHTSDO. The second keyword should be the process the document is related to, e.g. a Meeting. The third keyword should be an object, e.g. an Agenda.

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Editor</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.03</td>
<td>20090320</td>
<td>Eric Browne</td>
<td>Initial import from Framemaker</td>
</tr>
<tr>
<td>0.04</td>
<td>20090420</td>
<td>Eric Browne</td>
<td>Reorganisation - smaller topic files</td>
</tr>
</tbody>
</table>

© 2007-2009 International Health Terminology Standards Development Organisation. All rights of exploitation in any form and by any means are reserved worldwide for IHTSDO Members.

SNOMED CT was originally created by the College of American Pathologists.

This document forms part of the International Release of SNOMED CT distributed by the International Health Terminology Standards Development Organisation (IHTSDO), and is subject to the IHTSDO’s SNOMED CT Affiliate Licence. Details of the SNOMED CT Affiliate Licence may be found at http://www.ihtsdo.org/our-standards/licensing/.

No part of this document may be reproduced or transmitted in any form or by any means, or stored in any kind of retrieval system, except by an Affiliate of the IHTSDO in accordance with the SNOMED CT Affiliate Licence. Any modification of this document (including without limitation the removal or modification of this notice) is prohibited without the express written permission of the IHTSDO.

Any copy of this document that is not obtained directly from the IHTSDO (or a Member of the IHTSDO) is not controlled by the IHTSDO, and may have been modified and may be out of date. Any recipient of this document who has received it by other means is encouraged to obtain a copy directly from the IHTSDO, or a Member of the IHTSDO. (Details of the Members of the IHTSDO may be found at http://www.ihtsdo.org/members/).

Purpose of this document

This document is the introduction and overview of the IHTSDO Workbench, which forms the development and management environment for the content of SNOMED CT and related terminology products.

Status
The document is a working draft.

Source documents

Significant parts of this document were derived from "ACE Guidebook", the operations guide for the ACE Toolkit developed by Informatics Inc.
Overview of the IHTSDO Toolkit

The IHTSDO Workbench Toolkit constitutes a powerful and flexible environment for building, packaging and deploying tools for the production and management of clinical terminologies and classifications.

The toolkit comprises:

- a set of pre-built components - jar files
- a set of source code components
- a set of terminology data files
- a repository for distribution and management of components
- a set of scripts for automating various processes for building and deploying terminology development tools and associated artefacts
- a distributed workflow management system
- an extendable terminology editor
- an extendable terminology viewer

An IHTSDO IDE application bundle is a package of components ready for installation on an end-user's computer. Such a bundle includes the various software components, configuration files, policy files, predefined workflow tasks, a database containing one or more versions of SNOMED, and other additional term sets relevant to the terminology modeller's functional requirements. A terminology modeller may have several different bundles on his/her machine, particularly during a period of rapid functional change. In such cases, it is important to differentiate and clarify the functionality provided by each bundle.
IHTSDO Workbench Architecture

IHTSDO Workbench is founded upon a loosely-coupled distributed architecture that provides:

- Reliable operations over unreliable networks
- Business-process driven activities that can function reliably (and with high performance) while disconnected from the network
- No single point of system failure

Such loosely-coupled distributed architectures are common in applications where a rich user experience is required, where user productivity is a primary concern, and where disconnected operations are required. Some well-known applications that use these same architectural principles include e-mail and calendar applications, and most software development environments—such as Eclipse.

The IHTSDO Workbench architecture can scale computationally and organizationally as the internal needs of the organization expand, and as terminology collaborations expand to include an increasing number of external organizations.

Non-Proprietary Foundation

IHTSDO Workbench is based on a non-proprietary foundation. This non-proprietary foundation enables multiple vendors to provide tools and services. The non-proprietary foundation is based on a non-proprietary interchange format. This interchange format is based on the SNOMED interchange format but with added extensions to support the needed functionality.

This non-proprietary interchange is essential to remove and prevent vendor lock. This non-proprietary foundation is the basis for an extensible infrastructure, and is extensible through data access or extensions to the build environment.
Best-Practice Based

Develop Iteratively

Given today's sophisticated systems, it is not possible to sequentially first define the entire problem, design the entire solution, build the software and then test the product at the end. An iterative approach is required that allows an increasing understanding of the problem through successive refinements, and to incrementally grow an effective solution over multiple iterations.

The IHTSDO Workbench supports an iterative approach to development that allows developers to address the highest risk items at every stage in the lifecycle, significantly reducing a project's risk profile. This iterative approach helps attack risk through demonstrable progress—frequent, executable releases that enable continuous end-user involvement and feedback. Because each iteration ends with an executable release, the development team stays focused on producing results, and frequent status checks help ensure that the project stays on schedule. An iterative approach also makes it easier to accommodate tactical changes in requirements, features or schedule.

Manage Requirements—The IHTSDO Workbench provides the necessary tools to elicit, organize, and document required functionality and constraints, track and document tradeoffs and decisions, and easily capture and communicate business requirements.

Use Component-Based Architectures—The IHTSDO Workbench provides a framework that enables early development and baselining of a robust terminology architecture, prior to committing resources for full-scale development. It describes how to design a resilient architecture that is flexible, accommodates change, is intuitively understandable, and promotes more effective reuse.

Verify Quality—The IHTSDO Workbench allows terminology developers to build quality assessment into the process, in all activities, involving all participants, using objective measurements and criteria, without being treated as an afterthought or a separate activity performed by a separate group.

Control Changes—The IHTSDO Workbench allows users to manage change—making certain that each change is acceptable, and being able to track those changes. The IHTSDO Workbench provides the ability to control, track and monitor changes to enable successful iterative development. The IHTSDO Workbench also makes it simple to establish secure workspaces for each developer by providing isolation from changes made in other workspaces and by controlling changes of all artifacts (e.g., models, code, documents, etc.). And it brings a team together to work as a single unit by providing automated integration and build management.
The IHTSDO Workbench provides proven support for IDE Integration. There are many tightly integrated software IDEs such as the Eclipse IDE, which we use to develop the IHTSDO Toolkit.

In addition to software IDE integration, the IHTSDO Workbench Framework integrates into the software configuration management functions of the IHTSDO Workbench, and extended integration into the issues management, server management, and other collaboration tools is supported through the CollabNet Integration Framework.
The CollabNet Integration Framework (CIF) provides programmatic access to functionality and data within the CollabNet Platform. The CIF allows CollabNet customers and partners to build integrations and extensions that support application lifecycle management using the CollabNet Platform.

The IHTSDO Workbench is hosted using CollabNet SourceForge and CUBiT software. Both of these environments have fully functional web-browser interfaces, however in some cases, a tighter integration is desired. IDEs can integrate directly with these environments as shown in and .
Eclipse IDE

The CollabNet Desktop - Eclipse Edition, eliminates the need for developers to leave their preferred development environment and switch to other tools and systems to collaborate with remote colleagues and contribute to the lifecycle processes mandated by the enterprise. Eclipse users to connect to CollabNet CUBiT and access a centralized pool of build and test servers that can be dynamically allocated to their projects. Developers can now easily fix, rebuild and test code components, and track and manage efficiently the application configurations used for each build without leaving Eclipse.

CollabNet Desktop users can access their CollabNet Subversion repositories and perform all Subversion-related activities directly from the graphical Eclipse client. In addition, the CollabNet Desktop includes a powerful merge client that takes advantage of the merge tracking features that are available with CollabNet Subversion.

Users can interact with issue trackers in CollabNet SourceForge Enterprise directly from the Eclipse client, and they can access their CUBiT manager node and dynamically provision and access hosts. In addition, they can access all CUBiT functionality directly from Eclipse, such as assigning profiles, monitoring activity etc.

Terminologist IDE

The Clinical Terminologist’s IDE can be integrated into the CollabNet platform in an equivalent way that software developer’s IDEs are integrated. Today, the integration supports integration with the configuration management functions of the CollabNet platform, but future efforts could integrate the Clinical Terminologist’s IDE into the issue and project tracker functions of the platform as well. Such integration would allow for integration of the
Clinical Terminologist IDEs with trackers that might manage new term submissions, defect identification, and project management.

A prototypical Clinical Terminologist IDE is shown in . Within this IDE, the terminologist can view content via taxonomy navigation and advanced search, add new or edit existing content, develop business processes to automate batch activities, create maps from one terminology or classification to another, develop reference sets, manage dialects, review historical content, and complete scripted assignments delivered to them in an inbox.

This basic Clinical Terminologist IDE has been used, and customized by several organizations to meet their development needs.

Australia’s National Center for Classification in Health (NCCH) has taken the Clinical Terminologist’s IDE and have been using it to develop mappings between classification systems and SNOMED, and have also been creating SNOMED and classification system extensions using the IHTSDO Workbench IDE.

An example of how NCCH is using the IHTSDO Workbench IDE is shown in . In this Figure, note that the Emergency Department (ED) concept “Blunt injury – heel” shows two relationships of type “map to extension.” In this example, the original ED term and meaning needed to be preserved, but there was no existing SNOMED CT concept that exactly captured the meaning. Two modelers independently created a new SNOMED CT extension concept “Blunt injury of heel,” and used the relationship type “map to extension” to link the original ED term with the new SNOMED CT concept. Since both modelers independently created the map, and the extension concept, the NCCH extension now has a ‘duplicate’ that is easily removed via an IHTSDO Workbench business process that they created.
In this example, they have also demonstrated that their modelers can create mapping and extension content understandably, reproducibly, and usefully (URU) via a dual modeling and mapping.

NCCH has also been using the IHTSDO Workbench IDE to track mapping progress and show example metrics that they have been derived from the IHTSDO Workbench Environment.
Translation IDE

The IHTSDO Toolkit provides a strong foundation for a Translation IDE, and for managing dialects. Based on Java, the IHTSDO Toolkit provides full support for internationalization based on support in the Java SE Platform.

Like the Clinical Terminologist’s IDE, the Translation IDE can be integrated into the CollabNet platform to provide configuration management, issue tracking, and integrated project management capabilities.

Multi-Lingual Rendering Support

Core Java provides the foundation for internationalization of desktop and server applications. Java provides a development framework that encompasses:

- **Text representation:** the Java programming language is based on the Unicode character set, and several libraries implement the Unicode standard.
- **Locale identification and localization:** Locales in the Java SE platform are just identifiers that can be used to request locale specific behavior in many different areas of functionality. Localization is supported at the most basic level by the ResourceBundle class, which provides access to locale specific objects, including strings.
- **Date and time handling:** The Java SE platform provides various calendars, and supports conversion to and from calendar independent Date objects. All time zones in the world are supported.
- **Text processing:** The Java SE platform includes character analysis and case mapping, string comparison, breaking text into words and lines, as well as formatting numbers, dates, and time values into strings or parsing them back from strings. Most of these functions are locale dependent.
- **Character encoding conversion:** The Java SE platform supports converting text between Unicode and other character encodings when reading incoming text from streams or writing outgoing text to streams.

The user interface libraries in the Java SE platform enable the development of rich interactive applications. Internationalization aspects include:

- **Text input:** is the process of entering new text into a document - in the simplest case through typing on a keyboard, but often involving front-end software such as input methods, handwriting recognition, or speech input.
- **Text display:** is a multistep process that includes selecting a font, arranging text into paragraphs and lines, selecting glyphs for characters or character sequences, and rendering these glyphs. Some writing systems require bidirectional text layout for Arabic and Hebrew, or complex character-to-glyph mappings for Arabic, Thai, and the scripts of India. Text display is handled by the Java 2D graphics system and the Swing toolkit for lightweight user interface components and by the AWT for peered user interface components. The IHTSDO Workbench Toolkit utilizes the Swing toolkit and thereby extends Java’s full support for alternative glyph rendering.
- **User interface layout:** needs to accommodate text expansion or shrinkage caused by localization, and match the direction of the user’s writing system. For example, English-speaking users expect the tools in a tool bar to be...
organized from left to right, but users of a right-to-left language such as Arabic expect the tools to be organized from right to left. The IHTSDO Workbench Toolkit utilizes Java’s dynamic layout features, and can respond appropriately to alternative layouts.

End-User IDE

The IHTSDO Workbench IDE can also be configured for wide distribution to end-users. For example, the Australian Medicine’s Terminology (AMT) v1.4 (and later) viewer is available in a downloadable form from a secure NEHTA website. The data bundled with the viewer contains SNOMED CT identifiers and therefore requires interested stakeholders to have licensed SNOMED CT through NEHTA prior to obtaining it for review.

The AMT Viewer has all editing functions disabled, but if desired, edit functions could be enabled to allow for automated terminology submissions.

Even without editing functionality enabled, End-User IDE’s could have business process plugins data collection and automated submission for defect reports, or new term requests. These automated submission processes are transport layer independent, and can therefore integrate with receiving systems that use Web Services, Java RMI, SMTP, or Subversion as their transport layer.
The IHTSDO Toolkit provides two software frameworks for terminology-focused applications: one for build-process automation, and another for interactive development environments. These frameworks provide application skeletons that can be customized by an application developer to meet the unique needs of their end users.
The Toolkit Frameworks

The IHTSDO Toolkit provides two software frameworks for terminology-focused applications: one for build-process automation, and another for interactive development environments. These frameworks provide application skeletons that can be customized by an application developer to meet the unique needs of their end users.

Like software libraries, software frameworks aid the developer by providing software that solves problems for a given domain, and provides a simple API. However, while a library acts like a servant to other programs, a framework reverses the master/servant relationship. This reversal, called inversion of control, expresses the essence of software frameworks.

**Build Process Automation (BPA) framework**

The first framework — the IHTSDO Build Process Automation (BPA) framework — is based on the Maven tool for building and managing any Java-based project (The Apache Software Foundation, 2008). This BPA framework automates a build and management process that encourages use of industry best practices. Because of this foundation, the IHTSDO Toolkit:

- **Makes the build process automated and easy:** The BPA framework provides a declarative means for identifying project components, thereby simplifying the build system management. This declarative orientation provides the basis for the inversion of control of the build process automation framework. By making the build process automated, the build can be performed on a variety of continuous integration platforms.
- **Provides a uniform build system:** The BPA framework allows a project to build using its project object model and a set of plugins that are shared by all projects—software and terminology alike—providing a uniform build system. Once developers learn how one project builds, they automatically know how all projects build, saving immense amounts of time when trying to navigate many projects.
- **“At regular intervals, the process of “continuous integration” yields executable releases that grow in functionality at every release. # It is through these milestones that management can measure progress and quality, and hence anticipate, identify, and then actively attack risks on an ongoing basis.”**
  - Grady Booch
- **Provides quality project information:** The BPA framework provides project information reports that are in part taken from the project’s declarative definition, and in part generated from project sources. For example, the BPA framework can provide: change log documents created directly from source control systems, cross referenced sources, direct and transitive dependency reporting, and unit test reports including coverage.
- **Provides guidelines for best practices development:** The BPA framework aims to gather current principles for best practices development, and make it easy to guide a project in that direction. For example, specification, execution, and reporting of unit tests are part of the normal build cycle. Current unit testing best practices were used as guidelines. The BPA framework also assists in project workflow such as release management and issue tracking.
- **Allows transparent migration to new features:** The BPA framework provides an easy way for clients to update their installations so that they can take advantage of any changes that been made to the framework itself. Installation of new or updated plugins from third parties or the framework itself has been made trivial for this reason.

**Smart Client applications support work offline and they can be deployed and updated in real time over the network from a centralized server.**

The IHTSDO BPA framework builds on the Maven foundation by providing terminology-specific functions to manage, processes, and report terminology and classification data dependencies within JAR files, thereby providing a uniform framework for managing software and terminology dependencies.

**the IHTSDO Workbench Interactive Development**

The second framework—the IHTSDO Workbench Interactive Development Environment (IDE) framework—provides for high-performance end user applications, and uses a Java Swing-based framework that is easily extended—and scripted—using plain-old java objects. Both of these frameworks depend on lower level frameworks, interfaces, and libraries as shown in Figure 10, “Dependencies and layers of primary subsystems in the IHTSDO Toolkit”, on page 30.
Both the BPA based applications and the IDE based applications can be deployed either stand alone, or as Smart Clients applications. Smart Client applications can be run from the Internet, an Intranet, a network share drive, or on the workstation itself. Smart clients do not have the interface limitations of a webpage, and require no installation or worrying about having the current version because components can be downloaded automatically as needed.

Smart clients make sense because they enhance productivity as a result of their clear speed advantage, a richer, easier-to-use and manipulate interface, and the ability to work seamlessly offline, such as on an airplane. Economically, smart clients are no more expensive to manage than pure browser applications.

**Version-Oriented Terminology Interface**

Terminologies are by nature dynamic: new thing are added, old things are removed, and existing things are changed. The IHTSDO Toolkit supports this dynamic nature at a foundational level: it provides an explicit, comprehensive, uniform, and multi-dimensional representation of change across all aspects of the terminology content: concepts, descriptions, relationships, reference sets, reference-set specifications, and other terminology metadata.

The Version-Oriented Terminology Interface defines:

The rudimentary components (or building blocks) of the terminology (concepts, descriptions, relationships, reference set members, and reference-set specifications).
An immutable, or unchangeable, portion of each component. This immutable portion cannot be changed without fundamentally altering the identity of the component. For descriptions, the immutable portion contains the concept identifier and the description identifier of the description component.\footnote{1}

A mutable, or changeable, portion of each component. This mutable portion can be changed subject to permissions, and editorial policy. For descriptions, the mutable component includes the case sensitivity flag, the status, the language code, and the description itself.

A record of the time and path that uniquely identifies by whom and when a change is made.

An append-only representation of change, assuring that an audit trail is available for all changes to the terminology.

This explicit and comprehensive representation of change enables terminology developers to implement revision control and configuration management best practices.

This foundation enables distributed and collaborative development of terminology content, and is the lowest level dependency of the IHTSDO Toolkit. This foundation provides an explicit and consistent programming model for change, enhancing the developer’s ability to ensure that change is properly presented and managed.

**Change-Set Smart Client Model**

IHTSDO Workbench data architecture is founded upon an append-only data model where data changes are represented and distributed via a store-and-forward change set model. This append-only foundation eliminates the burdens of transaction isolation encountered in single database resources (since updates to the database are not allowed) and enables concurrent work. The change set model enables enhanced audibility, traceability, and data recovery.

The change set model also enables collaborative work via transport-layer independent coordination of change-set files.

IHTSDO Workbench uses the Subversion HTTPS protocols for the transport layer secondary to the simplicity and scalability of using Subversion for both the transport layer and for the revision control of the change-set files. In addition, since Subversion offers many types of end-user clients, the change sets can easily be exported or distributed by other means (FTP, SMTP, CD-ROM, network drive, etc), and still easily imported and integrated into a central Subversion repository.

The contents of the change sets are easily viewed by importing change sets into the IHTSDO Workspace Editor, where a complete history of all changes can be viewed in a graphical environment. Alternatively, change sets can be processed by other means and loaded into alternative database representation when required.

The loose coupling between the change set architecture and the underlying database allows independent migration of the database and of traceability capabilities, thus enabling support for alternative database schemas simplifying system evolution, and providing an opportunity for iron-clad traceability and non-repudiation via change-set enhancement that may span organization and technology boundaries.

Changes to the underlying database are based on the change set configuration management model. A baseline of data is established, data may not be deleted only changed, and changes are stored as deltas that describe two revisions of a given set of data – known as change sets. Therefore the baseline data plus all the change sets equals the current data state.

This model provides the ability to trace all changes made to the data over time, reliably rebuild the data to a specific point in time and easily see the state of the data at a given point in time.

The configuration management model is followed further within the application by providing each user a branch as a way of separating the content developed by one user from another. Each branch (known as a “path” in the IHTSDO Toolkit) has a baseline or “origin”, which may be another path or the root database, and its own set of change sets.

When a user synchronizes with the Subversion repository (performed regularly when the application is on-line) any new change sets created will be uploaded to the central repository and ultimately distributed to all other nodes. Equally any new change sets uploaded by other nodes will be downloaded and imported into the local database on the path to which the change sets belong.
IDE Version Control Best Practices

The IHTSDO Workbench IDE Framework is specifically designed to enable projects to implement industry best practices for version control within a smart client application. It aims to follow in the footsteps of software IDEs—such as Eclipse—that also enable projects to implement these industry best practices.

The IHTSDO Workbench IDEs are where clinical terminologists edit terminology, create mapping between terminology systems, translate descriptions, generate reference sets, classify the descriptions logic, and visualize, test, and debug what they or others have built.

The IHTSDO Workbench IDEs correspond to the notion of configuration managed workspaces or “sandboxes.” The IDEs allow the clinical terminologist to check out version-controlled components from a central repository (Subversion), and then record all the changes to those components with the intent of subsequently checking those changes back into the central repository.

The best practices for managing these IDE workspaces include (Wingerd & Seiwald):

Don't share workspaces. A workspace should have a single purpose, such as an edit/build/test area for a single clinical terminologist, or a build/test/release area for a product release. Sharing workspaces confuses people, just as sharing a desk does. Furthermore, sharing workspaces compromises the systems ability to track activity by user or task.

Don't work outside of managed workspaces. Each instance of an IHTSDO Workbench IDE is a loosely coupled managed workspace. Each IDE will generate change sets for all activity by default. These change sets can be coordinated in a transport independent and ad-hoc manner, or can be pre-coordinated on a project basis through use of managed accounts on a Subversion server. This flexibility allows for spontaneous collaboration without imposing a rigid structure on activities where such structure is not necessary or appropriate, while also allowing for strict project control when a more structured approach is desired.

Don't use jello views. Data in the IDE workspace does not change unless the clinical terminologist explicitly causes the change. A "jello view" is a workspace where data changes are caused by external events beyond the terminologist’s control. A typical example of a jello view is a workspace built upon a shared database where many users concurrently edit the same content. Jello views are a source of chaos in development. IHTSDO Workbench IDEs give the terminologists control over when their data changes.

Stay in sync with the baseline. The quality of a terminologists work depends on how well it meshes with other peoples' work. As changes are checked into the baseline, terminologists should update their workspace and integrate those changes. The IHTSDO Workbench IDE makes this workspace update operation straightforward and unencumbered with tricky or time-consuming procedures. Business processes that are configured by the IHTSDO Workbench IDE developer coordinate this update process.

Check in often. Integrating development work requires terminologists to check in their changes as soon as they are ready. The IHTSDO Workbench IDE allows configuration of startup processes, shutdown processes, and on demand processes, that encourage frequent check-ins.

Distributed Workflow Automation Layer

Coordination of activities is critical within and between organizations that want to work together to develop SNOMED CT.

IHTSDO Workbench provides a JavaBeans based Distributed Workflow Automation (DWFA) framework that is open and extensible, that has been developed specifically to support loosely coupled activities that occur when disconnected from the network, yet that can also coordinate with others via transport layer independent means.

As the number of participants in an activity grows, the resources required for coordination—and the risks associated with failure of coordination—may overwhelmingly increase. Workflow automation can simplify coordination of activities across time and space, enabling productivity increases and resource sharing that is not otherwise possible.

“By closely combining the unprecedented information communication capabilities of the Internet with the strategic business process automation and integration capabilities of workflow [...], significant changes will
Using workflow automation can increase productivity and improve the quality of SNOMED CT development by ensuring that tasks are automatically dispatched to the right worker, and by providing detailed reports of activities in progress, their status, and the workers responsible.

Workflow is the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules (Workflow Management Coalition 1999).

Using workflow, IHTSDO Workbench enables increased productivity and improved quality of operations by ensuring that tasks are automatically dispatched to the right worker at the right location, and by providing detailed reports of activities in progress, their status, and the workers responsible. Processes that have completed will report the statistics necessary to analyze workloads, costs, quality, and efficiency of operations, and the results of this analysis can be rapidly transformed into redesigned processes, resulting in higher-quality and greater efficiency.

Workflow processes form a major portion of the application functionality providing:

- scripted interaction with the end user
- binding from system processes to human interaction

The intent of the workflow system is to guide the user interaction to improve data quality and control quality processes such as dual independent review.

Workflow processes are an aggregation of tasks, which can be performed and configured graphically. Once constructed and configured, these processes are stored in a file, synchronized to the repository, and distributed via Subversion synchronization to all other nodes. Workflow processes are deserialised and executed on the target user's machine when the user selects to execute the task from their inbox queue.

Workflows tasks are the building blocks of workflow processes. Tasks interact with the IHTSDO Workbench API for user prompting and input as well as access to the underlying data store. Tasks may also gain access to panes in the IHTSDO Workbench GUI with full access at a Swing API level to perform any action required.

The relatively tight interaction between the workflow processes and the IHTSDO Workbench API gives a high degree of flexibility and richness for controlling interaction with the user.

A feature built into the workflow processes is an “aging queue”. Processes can therefore be delivered and not execute for a specified amount of time. This enables, among other things, setting a timeout for execution of a workflow process by creating a task on the aging queue to intervene if the workflow has not executed within a specified time period.

Workflow queues are used to assign work to individual users and track incomplete work. These queues are implemented as version controlled directories containing serialized JavaBeans that are synchronized via Subversion.

When required for execution a process is deserialized from the queue and executed. Storage of processes in queues or moving between queues is performed via an ACID compliant transactional API. Execution failure will cause rollback and reappearance of the workflow task.

Flow of Control

Flow of control determines which tasks in a business process are executed in which order. The IHTSDO Workbench DWFA framework has the following flow of control tasks:

- Branch Task: An unconditional branching into an arbitrary number of concurrent threads of execution.
- Conditional Branch Task: A conditional branching into an arbitrary number of concurrent threads of execution based on a conditional test. While the conditional branch function is generic in nature, the conditional tests themselves incorporate business logic, and therefore are implemented as subclasses of the Conditional Branch Task.
- End Branch: Unconditional end of a thread of execution.
- Synchronization Task: Provides a checkpoint to ensure that a specified set of concurrent threads has completed—and execution data from those execution threads have been collected—before continuing past the synchronization point.
point. This task works in coordination with the Check Synchronization Task, which is called by each concurrent thread of execution when it completes. This task allows for synchronization of an arbitrary number of concurrent threads of execution.

Check Synchronization Task: Determines if conditions of the Synchronization Task have been met, and updates the business process held by the Synchronization Task with any tasks completed and registered with the Synchronization Task.

Together, these flow-of-control tasks implement all the flow of control methods defined by the workflow management consortium. How the workflow management coalition defines these methods and how these flow of control methods are implemented in the DWFA framework, are presented in the remainder of this section.

Sequential Routing
A segment of a process instance under enactment by a workflow management system in which several activities are executed in sequence under a single thread of execution.

Sequential routing is the ordinary execution policy of workflow processes. All task are executed in a sequential order unless a Conditional Branch Task or a Branch Task is encountered during execution of a business process.

Parallel Routing
A segment or process instance under enactment by a workflow management system, where two or more activity instances are executing in parallel within the workflow, giving rise to multiple threads of control.

Parallel routing occurs whenever a Conditional Branch Task or a Branch Task that specifies two or more branches of execution is encountered during execution of a business process.

AND-Split
A point within the workflow where a single thread of control splits into two or more threads, which are executed in parallel within the workflow, allowing multiple activities to be executed simultaneously.

An AND-Split occurs whenever a Branch Task that specifies two or more branches is encountered during execution of a business process.

AND-Join
A point in the workflow where two or more parallel executing activities converge into a single common thread of control.

An AND-Join occurs when the Check Synchronization task confirms that all necessary conditions defined by the Synchronization Task have been met. Business process execution then continues using the thread of control that successfully performed the Check Synchronization task.

OR-Split
A point in the workflow where a single thread of control makes a decision about which branch to take when encountered with multiple alternative workflow branches.

An OR-Split occurs whenever a Conditional Branch Task—with a single branch for each possible value of the conditional test—is encountered during execution of a business process.

OR-Join
A point within the workflow where two or more alternative activity’s workflow branches re-converge to a single common activity as the next step within the workflow.

An OR-Join occurs when each of the possible branches of execution from an OR-Split, terminate with a Branch Task that all converge on a single task in the business process.

Iteration
A workflow activity cycle involving the repetitive execution of one or more workflow tasks until a condition is met.

Iteration is supported by a Conditional Branch Task that can branch to the beginning of the cycle until a conditional test is met and then when the conditional test is met, execution continues on one or more independent threads of execution.
**Transport Layer**

The business processes encoded by the IHTSDO Workbench DWFA Framework are JavaBeans, and can be transported in a transport layer independent manner.

The current implementation provides support for using Subversion, SMTP (as a MIME attachment), and Java RMI as the transport layer. Alternative transport layers that use MIME attachments and that have available Java implementations are trivial to support. Processes external to the DFWA framework can also support other transport layers such as FTP or CD-ROM combined with Sneaker Net or postal delivery services.

**Graphical Process Builder**

The IHTSDO Workbench IDE provides a graphical process builder for creating business processes, shown in . The graphical process builder enables processes to be created, saved, and restored. These processes can in turn be used as tasks in other processes allowing for data encapsulation and process reuse.

![Graphical Process Builder](image)

The availability of the process builder within the IHTSDO Workbench IDEs are configurable by the IDE developer. This configurability allows this functionality to be exposed to some users for some tasks, and also allows this functionality to be omitted when this functionality is inappropriate for the anticipated tasks or end users.

Business processes are created by dragging task components listed on the left hand side of the process builder to the process diagram on the right hand side, and then connecting the tasks together to form a defined sequence.

presents a prototypical dual-independent review workflow presented as a UML “swimlane” activity diagram. Workflows such as presented in that diagram can be implemented using the DWFA framework.
The graphical correspondence between the graphical process builder and the UML activity diagrams simplifies workflow implement from UML specifications.
User Inbox

The IHTSDO Workbench IDE provides a graphical user inbox for selecting and executing processes, shown in Figure 13: The IHTSDO Workbench IDE User Inbox.

Users can receive work in their designated in-box, and then execute them as appropriate. When executed, the process is removed from their queue.

Users also have access to a “read only” inbox where standard processes (such as a defect report process, or a new term request process) can be executed. In the case of , the read only inbox for an administrator is displayed, showing the process that will execute to add a new modeler to the system.

The IHTSDO Workbench centrally coordinates the user inboxes using Subversion—which also provides a recoverable audit trail of all activity within a user inbox.

In addition to central coordination, users can add processes directly to their inbox to personally manage appropriate types of work.

Inbox items can also be shared in a transport-layer independent and ad-hoc manner to facilitate spontaneous collaboration when appropriate.

Embedded Database

The IHTSDO Toolkit provides a high-performance embedded database implementation built on top of the Oracle Berkeley DB Java Edition. This Version Oriented Database (VODB) forms the primary data store for the IHTSDO Workbench IDEs. In addition, the same database engine can be used within a J2EE context to service other applications such as web-based thin clients for terminology viewing, searching, term submission, and lightweight editing activities.

Oracle Berkeley DB Java Edition is an open source, embeddable, transactional storage engine written entirely in Java. Oracle Berkeley DB Java Edition executes in the address space of the application, without the overhead of client/
server communication, allowing developers to store and retrieve information quickly, simply and reliably. Oracle Berkeley DB Java Edition was designed from the ground up in Java. It takes full advantage of the Java environment to simplify development and deployment. The architecture of Oracle Berkeley DB Java Edition supports very high performance and concurrency for both read-intensive and write-intensive workloads.

Relational databases are the most sophisticated tool available to the developer for data storage and analysis. Most persisted object data is never analyzed using ad-hoc SQL queries; it is usually simply retrieved and reconstituted as Java objects. The overhead of using a sophisticated analytical storage engine is wasted on this basic task of object retrieval. The full analytical power of the relational model is not required to efficiently persist Java objects. In many cases, it is unnecessary overhead. In contrast, Oracle Berkeley DB Java Edition does not have the overhead of an ad-hoc query language like SQL, and so does not incur this penalty.

The result is faster storage, lower CPU and memory requirements, and a more efficient development process. Despite the lack of an ad-hoc query language, Oracle Berkeley DB Java Edition can access Java objects in an ad-hoc manner, and it does provide transactional data storage and indexed retrieval, as you would expect from any database. The difference is that it does this in a small, efficient, and easy-to-manage package.

Oracle Berkeley DB Java Edition architecture is based on a log-based, no-overwrite storage system, enabling high concurrency and speed while providing ACID transactions and record-level locking. Oracle Berkeley DB Java Edition efficiently caches most commonly used data in memory, without exceeding application-specified limits. In this way Oracle Berkeley DB Java Edition works with an application to use available JVM resources while providing access to very large data sets.

In addition to supporting embedded applications, the Oracle Berkeley DB Java Edition fits into the J2EE architecture by implementing three key APIs within J2EE. By implementing the Java Transaction API (JTA), Oracle Berkeley DB Java Edition functions as a managed transactional resource within the application server. Oracle Berkeley DB Java Edition also implements the J2EE Connector Architecture (JCA) to ease integration into application servers. Finally, once integrated and performing transactional operations, most applications will require some ability to manage a service. Oracle Berkeley DB Java Edition exports information and services using the Java Management Extensions (JMX). In concert with JTA, JCA and JMX allow Oracle Berkeley DB Java Edition to operate to its fullest and in a predictable manner in J2EE-based solutions.

If an alternative database is desired, such an alternative can be easily integrated by implementing the Version-Oriented Database Interface shown in .

Lexographic Analysis, Search, and Spelling

The IHTSDO Workbench IDE Framework depends on a combination standard Java functionality (regular expression pattern matching and Unicode string representation), and Apache Lucene (Apache Software Foundation, 2008), for its lexographic analysis, search, and spelling functionality.

Lexographic Analysis

Apache Lucene performs considerable lexographic analysis as part of its index generation and query processing functions. These functions include extracting words, discarding punctuation, removing accents from characters, lowercasing, removing common words, reducing words to a root form (stemming), or changing words into the basic form (lemmatization).

The primary goal of this lexographic analysis is for information retrieval, not for automated machine translation. Independent of translation functions, Lucene can be configured differently for specific languages, different specialties, different acronyms, and for different abbreviations. Lucene currently offers support for a variety of languages.

Lucene offers multi-lingual support through its plug-in analyzers and stemmers.²

Lucene plug-in analyzers and stemmers include:

<table>
<thead>
<tr>
<th>English</th>
<th>Brazilian</th>
</tr>
</thead>
</table>

² [http://snowball.tartarus.org/](http://snowball.tartarus.org/)
In addition, Lucene provides other options for stemmers that use the Snowball analyzer.

: Not to be confused with SNOMED, Snowball is a small string processing language designed for creating stemming algorithms for use in Information Retrieval. #

Lucene Romance Languages Snowball stemmers include:

- French
- Spanish
- Portuguese
- Italian
- Romanian

Lucene Germanic Snowball stemmers include:

- German
- Dutch

Lucene Scandinavian Snowball stemmers include:

- Swedish
- Norwegian
- Danish

Other Lucene Snowball stemmers include:

- Russian
- Finnish
- Hungarian
- Turkish

Search

The IHTSDO Workbench IDE Framework provides three underlying search technologies that end users can select from.

The first technology is based upon Apache Lucene and provides a high performance—and indexed—search engine that uses a variety of lexical and morphological methods to provide rapid and effective search over language.

The second technology is based on Regular Expression pattern matching, and provides for unindexed pattern matching by iterating over all the descriptions in the database.

The third technology is a filtering capability that can be applied to the results of Lucene or Regular Expression queries, or can be applied directly to the database as independent queries.

Search by Lucene

Lucene is a high performance, scalable, information retrieval library (Gospodneti & Hatcher, 2005). Lucene can index and make searchable any data that can be converted to a textual format. Lucene does not care about the source of the data, its format, or even its language—as long as it converts to text.

Lucene enables IHTSDO Workbench IDEs to provide full text search capabilities that many databases don’t provide.

Lucene provides “sounds like” query capabilities through use of a plugin that implements the Metaphone algorithm. Lucene supports other plugin text analyzers and algorithms that can be used for other “sounds like” functionality in languages other than English. Alternatively, it may be of equivalent benefit to rely on inexact query methods that are not quite so language specific.

Metaphone is a phonetic algorithm, an algorithm for indexing words by their sound, when pronounced in English. The algorithm produces variable length keys as its output, as opposed to Soundex's fixed-length keys. Similar sounding words share the same keys. Metaphone was developed by Lawrence Philips as a response to deficiencies in the Soundex algorithm. It is more accurate than Soundex because it uses a larger set of rules for English pronunciation.
Lucene provides for other inexact query methods that allow for retrieval in circumstances when the spelling of a word is uncertain. One is through use of the Levenshtein distance algorithm\(^4\) during query formulation, which then enables searching the index for terms morphologically similar to the words of questionable spelling.

This use of an inexact query is automatic whenever a user puts a ‘~’ at the end of a questionable word—the user does not have to select the correct spelling from a provided list for the query to proceed. Instead a ranked list of matches to the inexact query is returned.

Lucene provides for Boolean queries using exact term or phrase matching, term distance criterion (how close two terms are to each other), similarity criterion, and wildcard criterion.

The Lucene searching capabilities in the IHTSDO Workbench IDE Framework provide simple to use yet powerful functionality for novice and advanced users.

**Search by Regular Expressions**

Regular expressions are a way to describe a set of strings based on common characteristics shared by each string in the set. They can be used to search, edit, or manipulate text and data.

The following examples illustrate a few specifications that could be expressed in a regular expression:

- The sequence of characters "car" in any context, such as "car", "cartoon", or "bicarbonate"
- The word "car" when it appears as an isolated word
- The word "car" when preceded by the word "blue" or "red"
- A dollar sign immediately followed by one or more digits, and then optionally a period and exactly two more digits

The regular pattern matching capability in the IHTSDO Workbench IDE Framework provides detailed pattern matching capability for intermediate and advanced users.

**Semantic Search**

Semantic search in the IHTSDO Workbench IDE Framework is implemented by an easily extended set of filters. These filters have complete access to the terminology database, and therefore can use any data in the terminology database as part of the filter criterion. Filters are based on edit date, status, parents, edit paths, and classification results-- and more are possible.

Filters can be applied against the entire database (find all concepts with a “finding site” of “cornea”), or can be applied to the results of a Lucene or regular expression search (final all concepts that contain “corneal” and something like “abrazion”, but that does not have a “finding site” of “cornea”).

Each of the semantic search filters allow the user to specify if they should include or exclude concepts that match the filter in the search results. Filters can be used multiple times in any order as shown in .

Currently, there are four semantic search functions that are meeting known current end-user requirements.

---

\(^4\) The Levenshtein distance is a metric for measuring the amount of difference between two sequences. The minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character, gives the Levenshtein distance between two strings.
Taxonomy Filter
The taxonomy filter tests for membership within a specified taxonomy.

Relationship Kind Filter
The relationship kind filter can test for:
The presence of a particular relationship type.
The presence of a relationship type that subsumes a particular relationship type.
The presence of a particular relationship type with a particular destination concept type.
The presence of a relationship type that subsumes a particular relationship type and a destination concept that subsumes another particular concept.
The presence of any relationship with a destination concept that subsumes any particular concept.
The presence of any relationship with a particular destination concept.

Refset Filter
The Refset filter tests for membership within a specified taxonomy.

Status Filter
The status filter checks for the presence of a particular status type on the concept of interest.

Spelling
Lucene provides a spelling checker to suggest a list of words similar to a misspelled word. This spelling checker uses a combination of Levenshtein distance and letter n-grams to suggest spelling alternatives.

The spelling checker is not currently integrated into the interface. We plan to initially make it available as a “data check” that will be applied as new descriptions are created, and applied again prior to commit, giving the user an opportunity to correct any errors.

Build Process Automation Framework
The build process automation framework is founded on Maven. Because of this foundation, we are able to leverage a large software development community and a build tool that provides the following benefits:
- Simple project setup that follows best practices - get a new project or module started in seconds
- Consistent usage across all projects means no ramp up time for new developers coming onto a project
- Superior dependency management including automatic updating and automatic dependency closures (also known as transitive dependencies)
- Able to easily work with multiple projects at the same time
- A large and growing repository of libraries and metadata to use out of the box, and arrangements in place with the largest open source projects for real-time availability of their latest releases
- Extensible, with the ability to easily write plugins in Java or scripting languages
- Instant access to new features with little or no extra configuration
- Model based builds: Maven is able to build any number of projects into predefined output types such as a JAR, WAR, or distribution based on metadata about the project, without the need to do any scripting in most cases.
- Coherent site of project information: Using the same metadata as for the build process, Maven is able to generate a web site or PDF including any documentation you care to add, and adds to that standard reports about the state of development of the project.
- Release management and distribution publication: Without much additional configuration, Maven will integrate with your source control system such as Subversion and manage the release of a project based on a certain tag. It can also publish this to a distribution location for use by other projects. Maven is able to publish individual outputs such as a JAR, an archive including other dependencies and documentation, or as a source distribution.
- Dependency management: Maven encourages the use of a central repository of JARs and other dependencies. Maven comes with a mechanism that your project's clients can use to download any JARs required for building your project from a central JAR repository. This mechanism allows users of Maven to reuse JARs across projects and encourages communication between projects to ensure that backward compatibility issues are dealt with.
Taken together, these features will allow IHTSDO and Release Centers to standardize and automate the terminology development process, resulting in increased productivity, responsiveness, and product quality.

### Terminologies and Classification Support

The IHTSDO Toolkit provides a framework for both terminology and classification development, without requiring any special “schema changes” to import terminologies of alternative structures.

We have already imported SNOMED CT, ICD-10-AM, ICD-9, ED (Australian Emergency Department Termset), CATCH (Australian Classification and Terminology of Community Health), and the Australian Medicines Terminology.

The IHTSDO Toolkit supports these alternative terminologies by using meta-data taxonomies to define their relationship, descriptions, and other attributes. The terminology or classification is then loaded into the embedded databases, and end-users can view, modify, and map between the terminologies and classifications within their environment as appropriate.

### Markup and Media Support

The IHTSDO Toolkit anticipates integration of standard xml, html, and multi-media capabilities into terminologies and classifications. Currently, the IHTSDO Toolkit supports description lengths in excess of 32,000 characters, supports storage of xml, rendering of xhtml and html, and the ability to associate images with concepts, and embedding those images in html markup. shows how images and xhtml are rendered in the IHTSDO Workbench IDE.
Images can also be associated with concepts, and designated as viewer images: images that will be displayed in the taxonomy view immediately prior to the description of the concept. shows several examples of concepts that have viewer images associated with them, and how these images are rendered in the IHTSDO Workbench IDE.

These viewer images can also be used in the display of “concept reference sets” as shown in of Reference Sets. In this case, the viewer image of the concept used in the reference set is displayed in the taxonomy view with the associated concept.
This dynamic association of viewer images with concepts allows reference set authors to configure how the reference set members are displayed in the taxonomy view.

**Reference Sets**

The IHTSDO Toolkit provides a generic and extensible model for representing reference sets. The model is based on separating the Refset data structure from the Refset purpose. Currently supported Refset types include:

<table>
<thead>
<tr>
<th>Membership</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Concept</td>
</tr>
<tr>
<td>String</td>
<td>Concept-Integer</td>
</tr>
<tr>
<td>Concept-Float</td>
<td>Language</td>
</tr>
<tr>
<td>Scoped Language</td>
<td></td>
</tr>
</tbody>
</table>

**Membership reference set**

A membership reference set is the parent or super type of reference set. This type of reference set purely defines membership, i.e., it can only collect members and, unlike other child formats, cannot attach additional values.

A membership reference set could be used to define a “formulary reference set”. Membership in this reference set defines that a medication is part of the particular formulary.

**Concept reference set**

The concept reference set extends the membership reference set by adding a concept value to the membership. This allows content developers to effectively add a concept value column to any component within the constraint of a reference set without having to alter the primary files.

A “medications schedule” concept reference set could indicate the sales restrictions on each medication. For each medication, this reference set could indicate a concept value indicating whether it is “unrestricted sale”, “pharmacy only”, “pharmacy restricted”, “prescription only” or “prescription only - controlled substance”.

**Concept-integer reference set**

The concept-integer reference set extends the membership reference set by adding a concept and integer value tuple to the membership. This allows content developers to effectively add concept and integer value columns as a bound pair to any component within the constraint of a reference set, without having to alter the primary files.

A "drug/disease interaction" reference set could use a concept integer reference set to link medications (e.g. beta-blockers) with diseases they might exacerbate (e.g. reactive airway disease), and use an integer value to indicate the severity of the interaction.

**Concept-float reference set**

The concept-float reference set extends the membership reference set by adding a concept and float value tuple to the membership. This allows content developers to effectively add concept and float value columns as a bound pair to any component within the constraint of a reference set without having to alter the primary files.

An “adult male reference normal” reference set could link a laboratory test result with a floating point number representing the average value, and a concept that represents the units of measure of the value.

**Integer reference set**

The integer reference set extends the membership reference set by adding an integer value to the membership. This allows content developers to effectively add an integer value column to any component within the constraint of a reference set without having to alter the primary files.

A "relationship order" integer reference set could be used to provide alternative ordering for displaying relationships in a hierarchical viewer for a selected concept(s) and its descendants.

**Boolean reference set**

The Boolean reference set extends the membership reference set by adding a Boolean value to the membership. This allows content developers to effectively add a Boolean value column to any component within the constraint of a reference set without having to alter the primary files.
A “drug availability reference set” indicating whether a drug is available nationally could be a Boolean reference set. A Boolean value of true would indicate that the drug is available, false that the drug is not available, and drugs not included in the reference set would have an unstated availability.

String reference set

The String reference set extends the membership reference set by adding a string value to the membership. This allows content developers to effectively add a String value column to any component within the constraint of a reference set without having to alter the primary files.

A String reference set could be used to annotate components with comments during a quality assurance or distributed development activity.

Specification Reference Sets

One use of the Concept Refset is to use as a “specification” for the generation of another. One advantage of this approach is that the specification of the reference set is strictly versioned using the same methods as the rest of the terminology and metadata content. Another advantage of using a specification is that it is easy to compute potential changes between one version of the terminology and the next.

Concept reference sets allow an icon to be associated with each concept associated with the reference set. shows the inclusion types for a particular type of reference set specification developed by the Australian National E-Health Transition Authority.
When a reference set specification is created, the taxonomy can display the viewer images of the reference set values associated with each concept. In , Fizzy drink is associated with a lineage include instruction, Milky drink is associated with an exclude lineage instruction, Cocoa drink is associated with an individual include instruction, Milk shake drink is associated with a lineage include instruction, Milky coffee is associated with an exclude individual instruction, and Sports drink is associated with an exclude individual instruction.

**Batch Editing**

The IHTSDO Workbench IDE Framework supports batch editing of collections of concepts. These concept collections can be generated in conjunction with search features, business processes, taxonomy includes, and individual addition or removal from the batch editing list. shows the batch editing list in the middle of performing a business process to move all the concepts from one parent to another.

Note that in the Figure, the user is being prompted to select the new parent for the list of concepts. After concept selection in the taxonomy view, and then pressing the "step" button, the business process running the batch operation will retire all the old is-a relationships for the concepts in the list, and add a new is-a relationship to the new parent.

The business processes that perform batch operations can be provided to the user, or developed by the user—depending on their skill and comfort with using the graphical process builder to create batch operations.

Business processes can “plug-in” to the environment complete with buttons and icons as also shown in . The icons on the right hand side of the list view are all business processes that have been placed in the plugins/list folder prior to environment startup. The button on the far right is the process that deletes old parents, and assigns a new parent to all the items in the list.
Batch editing business processes can be saved, restored, and shared in a transport independent manner. In situations where strict control over these business process plugins is required, the plug-in folder can be managed centrally using subversion, and startup processes can verify the integrity of the plug-in folder during startup.

Taxonomy Navigation and Display

The IHTSDO Workbench IDE Framework provides multiple visualizations to help users comprehend content that may contain multiple parents. For example, aspirin is both a salicylate product and an antiplatelet agent. It appears in the child hierarchies of both parent concepts. Concepts like aspirin that have more than one parent concept are said to have multiple lineage.

In the taxonomy view, concepts with multiple parents are indicated with a green up-arrow symbol, as shown in . Double-clicking on this symbol will display a list of the concept’s other parent concepts, as shown in . The up-arrow symbol will change to a diagonal arrow. Double-click this symbol to collapse the list.

In ,  and , three different visualizations for the concept “Amputation stump” are presented.

In , the standard taxonomy view is show with a second parent opened and expanded all the way back up to the top SNOMED CT Concept. This visualization uses a standard tree viewer as the basis of the representation, but expands the standard visualization by allowing all secondary parents to be expanded within the viewer— using different icons, and a different direction to help the user comprehend the multiple parentage being shown.
One advantage of this visualization is that you can view the siblings and children of the concept at the same time that the multiple parentages are shown.

shows a lineage tree rendering that only displays the multiple parentage, and pre-expands the tree to show the entire parent lineage for the focused concept. The focused concept appears multiple times in blue. It appears one time for each route between the focus concept and the top SNOMED CT concept. In this case there are two different routes, and thus the focus concept appears twice. In this representation the visualization is shown pre-expanded to the proper levels to help the user comprehend the lineage.

Figure 24, shows a lineage graph rendering that only displays the multiple parentages. An advantage of this approach is that the visualization often makes the lineage easier to comprehend, but the visualization is not as compact as the previous visualizations, and does not show siblings and children like the taxonomy view visualization does.
Change and Conflict Management

The IHTSDO Toolkit provides means to graphically present changes between any set of two or more development positions, and allows the user to visualize differences—and quickly resolve any differences if desired—within a simple interface.

If you have selected more than one view path, the Conflicts toggle displays (or hides) a view of conflicts between those paths. shows this interface populated with the shared components, the differences, and each of the original definitions of the concept “Trigger point (body structure)” in the 31-JAN-2003 release, and the latest SNOMED release.

The resolution panel on the upper left of the conflicts view shows the common (agreeing) components of the conflicting versions. The upper right panel shows the differences between the versions, and the bottom panels show the original definitions.

The color coding of the differences helps the user trace the differences back to their source.

Using this view, users can add components to the resolution by double clicking on them. The implement button will make the necessary changes on the current editing path to implement the resolution.
Data Validation

The IHTSDO Workbench IDE Framework enables structured data checks to occurring concept creation, and at commit time. These data checks can be made optional, and only provide a warning message, or they can be made mandatory and the user will be unable to commit changes while a particular data check fails.

Figure 26. shows an example of data checks that have identified unedited default values that are currently uncommitted. These data check warnings are presented without the use of modal dialogs, and are therefore less intrusive to the user. At commit time, if the data checks have not been resolved, a modal dialog is presented to the user, and the commit is unable to continue.
Customization

The IHTSDO Workbench IDE framework is designed from the ground up for customization. Developers can customize the environment by writing data check plugins, custom search filters, and custom business process tasks. These custom tasks that can script the environment, interact with the user, interact with the database, and provide alternative visualizations or custom functionality.

The CSIRO classifier integration is one example of a custom integration accomplished through the distributed workflow automation which later uses custom tasks.
The IHTSDO Workbench IDE and Viewer are quality assurance tools designed to help users develop standardized terminologies and/or multiple sets of the same terminology. Using the IHTSDO Workbench IDE, vendors can then incorporate standardized terms when developing products and services for clinics, hospitals and other medical organizations. Standardizing the terminology will bring consistency to diagnoses, patient records, insurance claims, etc.
Installing the IHTSDO Workbench IDE with Windows XP

To Be Determined
Using the IHTSDO Workbench IDE

The International Health Terminology Standard Development Organisation (IHTSDO) Workspace IDE supports agile methods for developing and maintaining concepts. The main window provides a number of features and functions to help you locate and modify concepts, as well as check for compatibility with other concepts.

Launching the IHTSDO Workbench
Using the Hierarchy View
Reviewing the Concept Work Area
Reviewing the Menu Bar
Understanding the Preferences Panel
Using the List View
Using the Search Function
Using the Search Variables
Executing an Advanced Search
Submitting a Change Set to the IHTSDO Workspace
Using Reference Sets
Adding a Concept to a Specification Reference Set
Displaying Specification Refset Concept Types in the Hierarchy List
Displaying Refset Members in the Hierarchy List
Changing the Refset Default Values
Setting Refset Preferences
Inclusion Types Inclusion Type
Conclusion
Launching the IHTSDO Workbench

The IHTSDO Workbench is launched from the self-launching icon, which can be found in the IHTSDO Workbench folder on your computer hard drive, or in the Start menu (on Windows-based PCs.) The IHTSDO Workbench icon is shown in .

Click (or double-click) on the icon to launch IHTSDO Workbench. The next screen will prompt you to select your user profile, as shown in .

Click on the “bootstrap.ace” or other appropriate profile and click Open. The Activity Viewer, which displays a graphic as the concepts and other system components are loaded, will display, as shown in .

When loading activity is finished, the Config dialog, shown in will display.
Enter your user name and password and click OK. The IHTSDO Workbench Editor window will display, as shown in

The Main IHTSDO Workbench IDE window contains these primary sections or areas:

Display Toggles: The buttons in the horizontal bar at the top of the window and within the Concept Work Area turn on and off functions and views of concept descriptions and data in the Concept Work Area. Toggles are universal to the configuration of the entire IHTSDO Workbench IDE window.

Business Process Plug-Ins: When invoked, these buttons launch business processes related to creating and changing concepts.

Hierarchy View: The horizontal section of the window provides access to the available hierarchies, a visual representation of relationships between concepts.

Concept Work Area: Tools to view details about and modify concepts.

Search Function Pane: The lower section of the window provides access to a search tool to help you locate the concept to be modified or used.

Reviewing the Top Row of Toggles

The top row of toggles provides the following functions. Some of these functions may be disabled depending on the IDE that is executing.
History Toggle
Displays or hides a list of committed and uncommitted concept changes.

Address Book Toggle
Displays or hides addresses of project participants.

Hierarchy Toggle
Displays or hides the Hierarchy View area and uses the entire width of the window to display the concept (as shown in the Concept Work Area), as shown.
Component Toggle
Displays or hides the hierarchy view of the concept as shown .

Synchronize with Subversion Business Process
If displayed, this button can execute a business process that will synchronize local data with a subversion repository.

Transporter Toggle
Provides a way to quickly populate one of the unassigned concept work area tabs, or to locate a concept in the hierarchy, using drag and drop.

Queue Viewer Toggle
Displays or hides queues from which users can launch processes, including work assignments. For more information about using queue viewer features, .

Process Builder Toggle
Displays or hides the Process Builder, where you can create or execute business processes using the concepts you modify, as shown . For more information about building processes, see Exercise - Creating a Business Process .
Preferences Panel Toggle
Displays or hides the available hierarchy versions and other settings. In most environments, the project manager will set up and maintain preferences for each user, including the edit path.

Using the Hierarchy View

The Hierarchy View includes two tabs: taxonomy and data checks.

The **Taxonomy** tab contains sets of concepts that are commonly used for patient diagnoses and records. The “pluses” to the left of each concept indicate that sub-levels, or children of the parent concept, exist. Click on a plus sign to view cascading sub-levels under each concept. For example, if you click on the “SNOMED CT Concept” lever, the list will expand as shown in .
Similarly, if you continue to click on the levers (or triangles) next to each sub-level, you can view additional levels of the concept. At the same time, you will note that the concept hierarchy information for the selected hierarchy displays in the Concept Work Area.

**Reviewing the Hierarchy View**

The arrows next to concepts in the hierarchy view denote relationships. The example shown includes several types of arrows, to indicate relationships within the hierarchy.

Levers are used to open and close parent and child views:

- **Parent/child view**: The lever’s (or triangle) toggling will show or hide the children of the parent.
- **Green vertical “up” arrow**: Opens and closes additional parents, as in “Reason not stated concept”.
- **Green angled arrow**: Indicates the “open” state of the vertical arrow.

A concept may appear in more than one place in the hierarchy. For example, aspirin is both a salicylate product and an antiplatelet agent. It appears in the child hierarchies of both parent concepts. Concepts like aspirin that have more
than one parent concept are said to have multiple lineage, and are indicated in the hierarchy view with a green up-arrow, as shown in .

Double-click on the green up-arrow to display a list of a concept’s other parent concepts, as shown in . The up-arrow will change to a diagonal arrow. Double-click this symbol to collapse the list.

**Using the Data Checks Tab**

When IHTSDO Workbench detects a data inconsistency in a concept, the data checks tab will automatically display an error message. If resolutions are available, they are displayed in the Fixes field, as shown in . When all errors are resolved, the screen will automatically display the taxonomy tab.

You may be able to submit a concept with data checks if the basis of the data check is permissive. You may be prompted to state whether you want to continue with the submit process.

**Reviewing the Concept Work Area**

The Concept Work Area provides an array of tools to assist you in adding and modifying concepts, modifying source and destination relationships and submitting change sets via the Subversion Panel.

**Reviewing the Concept Work Area Toggles**

The row of buttons (shown in ) immediately above the concept work area provides the following universal functions.
FIGURE 45. Concept Work Area Toggles

ID Toggle
Concept Attributes Toggle
Descriptions Toggle
Source Relationships Toggle
Destination Relationships Toggle
Lineage Toggle
Lineage Graph Toggle
Images Toggle
Conflicts Toggle
Refsets Toggle
Preferences Filter Toggle
Description History Toggle
Preferences Toggle
**ID Toggle**
Displays or hides the Universally Unique Identifier (UUID), and SNOMED ID of the selected concept, a record ID specific to the item on the screen. Also displays the UUIDs of the concept's associated descriptions and relationships.

**Concept Attributes Toggle**
Displays or hides the attributes or state of completion of the concept. Concept attributes include:

- Status of the concept (current, retired, etc.)
- Whether the concept has been fully defined (true/false). Most concepts are defined, and their defined attribute is set as true. However, there are abstract or theoretical concepts that are not defined, for example, the cornerstone concept at the top of a hierarchy. For such concepts, the value of the defined attribute should be set as false.

Click on the History toggle to view the status of each section of the concept in the work area.

**Descriptions Toggle**
Displays or hides all of the descriptions associated with a concept, whether they are synonyms or represent the preferred name, fully specified name, or other description type.

**Source Relationships Toggle**
Displays or hides the “is a” source relationships for which the selected concept is a parent. It also displays the other AMT and SNOMED-specific relationships for which this concept is the source.

**Destination Relationships Toggle**
Displays or hides the “is a” destination relationships for which the selected concept is the child. It also displays other AMT and SNOMED-specific relationships for which this concept is the destination.

**Lineage Toggle**
Displays or hides a visual aid of the lineage, or parent relationships, of the selected concept. The selected concept is designated with blue arrows to highlight where its multiple lineages, if any, exist.

**Lineage Graph Toggle**
Displays or hides access to the lineage graph tool, which provides a dimensional view of the concept currently displayed in the concept work area of the screen. The lineage graph toggle is controlled via a checkbox in the Component tab of the Preferences Panel.

**Images Toggle**
Displays or hides access to the image (such as a logo or photo) upload task associated with the selected concept.

**Conflicts Toggle**
Displays or hides the Conflicts section of the work area, where differences in concept definitions can be analyzed and resolved.

**Refsets Toggle**
Displays information about the reference sets associated with the selected concept.

**Preferences Toggle**
Changes the view to display preferences.

**Description History Toggle**
Provides a view of all states of the descriptions added to and removed from this concept and the path in which the changes were made.
Preferences Toggle
Uses specified preferences to filter the information displayed to the user.

New Concept Toggle
Creates a new concept as the child of the currently selected concept in the hierarchy viewer.

Concept Linkage Toggle (above Process Icons)

The Concept Linkage toggle or “spinner” determines whether the currently selected concept is linked to the hierarchy or some other view. Click on the spinner arrows next to the concept linkage icon to link and unlink the current concept from the hierarchy. You may link link any tab’s display to the concept selected in the hierarchy pane, the search window, or the data checks window; or you may choose nto to link to any window. Select the tab whose linkage you wish to change and then select a new linkage using the linkage spinner. Using the Concept Linkage toggle provides more control over the environment you are working in.

Reviewing the Menu Bar

The Menu Bar, as shown in , provides tools to manage certain application and configuration functions, as described below, that are typically handled by a system administrator. The functions available via the menu bar may vary from installation to installation, and may include the ability to invoke local business processes.

File Menu

* Export Baseline Jar - Uploads committed changes to subversion.
* Import Java Changeset - Downloads changes from subversion.
* Change Password - Allows you to change your logon password.
* Save Profile - Save the current IHTSDO IDE status (including taxonomy display, displayed icons and windows) in a profile, which is selected at startup. IHTSDO IDE provides another opportunity to save this profile when you quit the application.
* Save Profile As... - Allows you to save an alternate profile.
* Show Configuration Info - Displays the current version of the application database.
Quit - Exits IHTSDO IDE.

Edit Menu

Cut, Copy, Paste - Provides access to standard editing functions - Cut, Copy and Paste. You can also access these functions via keyboard shortcuts: Ctrl-X, Ctrl-C, Ctrl-V).

Collabnet Menu

IHTSDO Workspace - Advances the screen to the Informatics Collabnet site, where application development and changes are stored.

Window Menu

New - Provides options to create new instances of the Activity Viewer or Log Viewer.

Activity Viewer, User Profile, Workflow Bundle - Provides options to access different instances of IHTSDO Workbench.

Understanding the Preferences Panel

The Concept Work Area displays a concept view based on the concept descriptions you select from the Hierarchy View and those you retrieve from a search. The Preferences Panel provides multiple options for controlling the concept information and its presentation.

Using the Preferences Panel

The Preferences Panel controls the relationships displayed in the hierarchy view. Before you begin to construct or change concepts, you need to select the version of the hierarchy you want to view and edit. This selection will also determine the hierarchy views available to you.

To access Preferences, click on the Preferences Panel icon. A window similar to the one shown in will display. This example shows a path with multiple versions.

The presented view will depend on which path was selected, as shown in the available Path tab. To designate a path to use, you must use the “thermometer” slider to select a version, as indicated by the arrows to the left of the available choices.
**Using the Preferences Panel Tabs**

In addition to the Path tab, the Preferences Panel includes these additional tabs across the top of the panel.

- View
- Edit
- New Path
- Refset
- Component Panel

The *Preferences View tab* displays the current settings and options for descriptions, relationships, status and roots, as shown in .
**Descriptions**

*Description Types* - Specifies which taxonomy description types will be standard for the current settings, such fully specified name or preferred term.

*Short label preference order* - Specifies how to display the concept name in those areas of the screen that use short labels (e.g., the taxonomy pane), based on the concept description type. The first concept description whose description type matches one of the description types in the list (in descending order) will be the one displayed for that concept.

*Long label preference order* - Specifies how to display the concept name in those areas of the screen that use long labels (e.g., the concept work area), based on the concept description type (e.g., fully specified name, preferred term). The first concept description whose description type matches one of the description types in the list (in descending order) will be the one displayed for that concept.

*Tree preference order* - Specifies how to display the concept name in the taxonomy view pane, based on the concept description type (e.g., fully specified name, preferred term). The first concept description whose description type matches one of the description types in the list (in descending order) will be the one displayed for that concept.

*Table preference order* -

Roots - Defines the highest level (root) concepts in the taxonomy. All other concepts will be children or sub-children of these concepts.

Taxonomy

*Display toggles* - Checking one or more boxes will control how data is displayed in the taxonomy view

*Parent relationships* - Defines the relationships used to associate child concepts with parent concepts.

*Stated view characteristic types* -

*Inferred view characteristic types* -

Taxonomy Sort - If the Sort taxonomy using refset is selected on the Taxonomy tab, this sort defines the refsets that will order the taxonomy display.

Status - Defines which concepts will be displayed in the taxonomy view pane based on their value in the associated status field.

Refset - Specifies the refsets for which concepts in the hierarchy display will show adjacent inclusion or exclusion symbols.
The *Preferences Edit tab* displays the default settings for status, description, relationship type, relationship characteristic and relationship refinability. An example of default settings is shown in.

**Edit Defaults**

*Default status* - Specifies the initial value of the status field when a new concept is added.

*Default image type* - Specifies the initial value of the status field in the Image section of the work area when associating an image with a concept.

*Default description type* - Specifies the initial value of the description that will appear in the type field of the Descriptions section of the work area when a new concept is added.

*Default relationship type* - Specifies the initial value in the type field of the Source relationships and Destination relationships sections of the work area when a new concept is added.

*Default relationship characteristic* - Specifies the initial value in the char (characteristic) of the Source relationships and Destination relationships sections of the work area when a new concept is added.

*Default relationship refinability* - Specifies the initial value in the refinability field of the Source relationships and Destination relationships sections of the work area when a new concept is added.

Relationship Types (values for popup menu) - Defines the values available for the relationship type of a concept. These values specify the choices displayed in the refinability field popup selection boxes in the Source relationships and Destination relationships sections of the work area.

Relationship Refinability (values for popup menu) - Defines the values available for the refinability of a concept. These values specify the choices displayed in the refinability field popup selection boxes in the Source relationships and Destination relationships sections of the work area.

Relationship Characteristics - Defines the values available for the characteristics of a concept. These values specify the choices displayed in the char (characteristic) field popup selection boxes in the Source relationships and Destination relationships sections of the work area.

Description Types - Defines the values available for the description type of a concept. These values specify the choices displayed in the Descriptions section of the work area.

Image Types - Defines the values available for a concept’s image type. These values specify the choices displayed in the Images section of the work area.

Status Values - Defines the values available for the status value of a concept. These values specify the choices displayed in the status field popup selection boxes in the Concept attributes, Refset, Descriptions, Source relationships, Destination relationships, and Images sections of the work area.
You can add items to the defaults by copying and pasting from the Hierarchy View, as shown in . The added item(s) will then display in the popup menu available in the Concept Work Area.

The added item(s), as shown in , will then display in the popup menu in the Concept Work Area.
The Preferences New Path tab provides a way to create a new path (hierarchy view) from the existing paths for Terminology Auxiliary and SNOMED CORE taxonomies, as shown in.
To add a new path to the available tabs, as shown in , follow the instructions below.

Click on the New Path tab. Two fields will display at the top of the Path tab area: Description for new path and Parent for Path (which will contain the text “Empty” in red).

From the Hierarchy View, select the desired path.

Enter a description of the path in the Description for new path field.

Enter the correct parent path in the Parent for Path field OR click in the “Use position as origin” box in the space below the tabs to define the current position in the Hierarchy View as the parent path.

Click the Create button to add the new path to the available selections.

Preferences Panel - RefSet Tab

For information about reference set preference settings for the tabs shown in , see Displaying Specification Refset Concept Types in the Hierarchy List, and Displaying Refset Members in the Hierarchy List.

The result of a specification refset is a member refset, i.e., the set of concepts specified by the specification refset. IHTSDO IDE can be set to display which concepts are members of a particular member refset. A diagonal paper clip icon next to a concept means that the concept is a member of the specified refset. A horizontal paper clip icon next to a concept indicates that one or more of its children is a member of the refset. To display these icons, you must drag the desired member refset name onto the refset sub-tab.

Preferences Panel - Component Tab
The checkboxes on the Component panel tab determine which display toggle buttons are shown or hidden. Hiding a button ensures that the corresponding portion of the work area is either always displayed or always hidden (i.e., cannot be toggled on or off).

**Reviewing the Concept Work Area Tabs**

In addition to the icons described in *Reviewing the Concept Work Area Toggles*, this portion of the IHTSDO Workbench IDE window includes five tabs. The first four tabs in the concept viewing and editing area are user-configurable. By default, the first tab is linked to the current selection of the hierarchy view. However, the first tab can be “unlinked” by clicking on the Concept Linkage button.

In the example shown in , the Hierarchy tab contains the hierarchy view name and the Search tab displays “Search” since a concept search has not been executed.

**Hierarchy Tab**: Displays the selected concept’s descriptions and relationships from the Hierarchy View.

**Search Tab**: Displays the concept, if any, selected from a search.

**Empty and Empty2 Tabs**: These tabs are available for constructing new concepts by copying a concept, as shown in , and then changing their descriptions and relationships. Concepts can be copied by using the drag and drop feature of the Transporter icon. The *Empty and Empty2* tabs are not linked to the hierarchy.
**List Tab:** The List tab provides a way to execute batch changes such as to associate a group of children with a different parent. For more information about functions available from the List Tab, see *Using the List View*.

**Adding a Concept**

Concepts can be added by modifying an existing concept in the hierarchy tab view or copying a concept to one of the Empty tabs and making additions and changes from that view.

To add a concept, click on the plus sign + in the Descriptions icon of the work area, as shown in .

To create the new concept, double-click on the blank text field. The current focus concept text will fill in. Change the existing text to reflect the new concept. For more information about adding descriptions, *Adding and Changing Descriptions***

**Modifying a Concept**

To modify an existing concept, right-click on the concept description to be modified. A popup menu will display, as shown in .
Click on Change: [description]. A new blank line will appear below the line you want to change.

Double-click in the text field of the new blank line and the existing concept description will fill in. Place your cursor within the text field and modify the text.

For more information about changing concepts, *Adding and Changing Descriptions concept descriptions adding and changing*

**Changing Concept Attributes**

This section of the work area displays the existing concept and retired attributes and indicates whether the concept has been fully defined.

To change or retire a concept attribute, hover the cursor over the existing line. The Change or Retire menu displays. Click on one of the options. Note that the “status” column will show the option you selected. A new line will display. Click on the new line to reveal a True/False choice. Click on the appropriate choice.

*Green Bar:* Note that new lines that have not been committed to the database are identified by a green vertical bar at the left side of the line, as shown in . This visual identifier is used throughout the IHTSDO Workbench Editor to denote uncommitted additions. The green bar will remain in place until you press the “Commit” button.

### Adding and Changing Descriptions

This section displays all of the descriptions that either are synonyms for the selected focus concept or represent the preferred or fully specified description.

To change or retire an existing description, hover the cursor over the existing line. The Change or Retire menu displays. Click on one of the options. Note that the “status” column will show the option you selected. In either option, a new line will display, as shown in .

The focus concept is the first description listed in the section. To add a new description, click on the green plus sign under the “Descriptions” heading. A new line with the default text “New Description” will be added to the bottom of the list, with a green vertical bar on the left side of the line to indicate the line has not yet been committed to the database. The green bar will be displayed until you press Commit or Cancel. *Note: This identifier is used throughout the IHTSDO Workbench Editor to denote uncommitted additions.* Double-click on the text field in the new line and replace the existing text with the description you want to add to the list, as shown in .

The remaining fields are described below.

*Note: If you are editing on two paths (see Preferences Panel settings), clicking on the green plus sign will create two blank lines.*

**Type:** Type defaults to “synonym.” Click on the Type field to view a popup list of other options that include fully specified, xhtml def(inition), entry term, preferred and unspecified.
Case: The Case field denotes whether the concept, as a whole, is sufficiently defined. The default is false. You or the user must determine whether the concept is adequately defined. Click on the Case field to choose true or false.

Language (Lang): The default is “en” (English).

Status: The default is current. Click on the Change Status option in the drop-down list to view and select a different status description from the popup list. The choices that appear are set in the Preferences Panel.

Version and Path: These fields are displayed only when the “Display history records” button is pressed. These informational fields are automatically updated by the IHTSDO Workbench Editor.

**Adding and Changing Source Relationships**

This section displays all of the source relationships that are already defined for the focus concept. Relationships link concepts to one another. That means “is a” relationships link concepts in a parent-child hierarchy (e.g., duck is a bird; bird is a animal). A concept can have more than one “is a” relationship to other concepts (e.g., bird is a animal, bird is a verterbrate.)

Attribute relationships form a class of similar relationships that relate two concepts and establish the type of relationship between them. These types of relationships are defined by the designer of the hierarchy or database. For example, the attribute relationships in a pharmacology database could include “Has dose form” and “Has active ingredient”.

A relationship exists between two concepts. That relationship may appear as either a source relationship or a destination relationship, depending on which of the two concepts’ details you are viewing. Relationships with concepts higher in a hierarchy than the concept are source relationships. The IHTSDO Workbench Editor displays both source and destination relationships for a concept.

To change or retire a source relationship, hover the cursor over the existing line. The Change or Retire menu displays, as shown in . Click on an option. Note that the “status” column will show the option you selected.

To change a description, select the “Change” option.

To add a source relationship, click on the green + symbol under the “Source Relationships” description. A new line will be added to the list with the default relationship type “is-a rel (terminology constant)”. Copy and paste a description from the Hierarchy View, as shown in . Note: Added line has been enlarged in screen shot for emphasis.
Remaining fields in the Source Relationships section are described below.

**Destination:** The concept to which this concept is connected in the terminology hierarchy.

**Char:** Characteristic of the source relationship, as defined by options in the popup menu (displayed when you double-click on the column).

**Refinability:** Indicates whether the refinability of this relationship is optional, mandatory or not refinable, as defined by options in the popup menu.

**Group:** The SNOMED relationship group to which this relationship belongs.

**Status:** Indicates the concept description, as defined by the popup menu, e.g., current or inactive.

**Adding and Changing Destination Relationships**

The Destination Relationships section displays all of the destination relationships that are already defined for the focus concept and provides a way to change and add destination relationships.

To change or retire a destination relationship, right-click on the desired line and choose an option from the displayed popup menu, as shown in [image].

To change the content of a destination relationship, right-click on any of the fields and select an option from the displayed popup menu.

To add a destination relationship, click on the green + symbol under the “Destination Relationships” description. A new line will be added, as shown in [image]. Copy and paste or type the new destination relationship values.

Remaining fields in the Destination Relationships section include:

**Origin:** The source relationship of the description.

**Type:** Type of relationship, as shown when you triple-click on the field to display the popup menu.

**Char:** Characteristics of the destination relationship, as defined by options in the popup menu.

**Refinability:** Indicates whether the refinability of this relationship is optional, required or not refinable, as defined by options in the popup menu.

**Status:** Indicates the destination description, as defined by the popup menu.

**A Note About Taxonomy**

A relationship contains a taxonomy when a true “is-a” (source relationship) statement is included in the view. If the view only displays mapping to other concepts, the relationship is not a taxonomy.

**Viewing Concept Lineage**

The Lineage section of the Concept Work Area provides a visual aid to help identify where the selected concept resides in the hierarchy. The selected concept is designated with blue arrows to highlight where its multiple lineages, if any, exist, as shown in [image].
As described in the Hierarchy View discussion (Reviewing the Hierarchy View), the slanted green arrows indicate additional parent lineages in higher positions in the hierarchy.

**Using Concept Lineage Graphs**

As noted in Lineage Graph Toggle Lineage Graph Toggle togglelineage graph, you can view or hide access to a lineage graphing tool that provides dimensional views of the current concept’s lineage. The default setting for the icon is “off.” To turn the graphing option on, from the Preferences Panel, select Component Panel and click in the box next to Lineage Graph.

This tool is especially helpful when working with complex concepts. Lineage graphs can be manipulated according to a number of algorithms, allowing you to reorganize the lineage elements to suit your needs.

*Note: You may want to hide the sections of the concept work area above the lineage graph section to provide space for a larger display of the graph.*

To illustrate the features of the lineage graphing tool, use the search function to search for ‘duck viral hepatitis.’ This concept will yield a complex graph that can be reorganized by the lineage graph tools. The initially displayed graph will look like the one shown in . The focus concept element is outlined in blue; the top SNOMED CT concept element is outlined in red.
Lineage Graph Controls (Icons)

The icons on the left side of the lineage graph, as shown in , provide controls for viewing the graph. The controls provide the following functions:

Lens Views - Use this control to change the graph image to hyperbolic (a curved representation) views that are useful for large, complex graphs, as shown in , or a hyperbolic layout, as shown in that distorts the image of the elements. You can also magnify the view or layout from this menu. Click “No lens” to return to the original view.

**FIGURE 73. Lineage Graph Controls**

![Graphing Icons](image)
Magnify View - Click this icon to incrementally magnify, or enlarge, the graph view. You can also use the scroll wheel on the mouse to magnify a view. The cursor position determines where magnification begins.

Shrink View - Click this icon to incrementally shrink the graph view. You can also use the scroll wheel on the mouse to shrink a view.
Nodes View - Click this to enable use of the mouse to select nodes within the graph and move them around, as shown in . Use the mouse to draw a line around the nodes you want to move. Click in the middle of one of the selected nodes and drag to move the node. To extend the selected node beyond the original borders, Shift/Click in the middle of the node. To deselect the node, click outside the selection borders.

Algorithm View - Click this icon to choose a specialized view based on the algorithms listed in the menu. For more information, see Changing Lineage Graph Layouts with Algorithms graphsconcept lineage layouts.

Using Lineage Graph Keyboard Controls

In addition to the controls/icons discussed above, lineage graphs can also be manipulated via several keyboard and mouse controls.
- Click/Drag - moves the graph within the graph window
- Shift/Click - rotates the graph
- Ctrl/Click/Drag - (or Clover/Click/Drag for Mac users) - when positioned in the center of the frame, moves the graph from side to side, as a shear function
- Mouse Hover - displays a tool tip that is the selected graph element’s label (e.g., V2)
- Click a Link - changes the line to red to show where the line leads, i.e., to illustrate relationships within the concept.

Changing Lineage Graph Layouts with Algorithms

As noted in Lineage Graph Controls (Icons) graph controls, the Algorithm View icon provides access to a number of graph views based on algorithms. The algorithms, by definition, incorporate some randomness that can change how the layout appears in different instances of the same algorithm.

Each algorithm has advantages and disadvantages. Some are more suited to small, simple graphs; others work best with larger, more complex graphs. The available algorithms are described below.

**DAG Layout** - is best suited to tree-like directed acrylic graphs. The layout will have directed edges generally pointing upwards. Any vertices with no successors are considered level zero, and tend toward the top of the layout. Any vertex has a level one greater than the maximum level of all its successors.

**ISOM Layout** - implements a self-organizing map layout algorithm, based on Meyer’s self-organizing graph methods.

**FR Layout** - implements the Fruchterman-Reingold algorithm for node layout. This is one of the two most widely used layouts, and is based on the spring algorithm, which introduces attraction forces between linked nodes and
repulsion forces between unlinked nodes. The FR layout uses a “temperature” system to lay out the entire graph. During each iteration of the algorithm, the temperature is reduced. When the minimum temperature is reached, the algorithm stops.

**KK Layout** - implements the Kamada-Kawai algorithm for node layout, also a widely used layout scheme. Each iteration of the KK Layout changes the position of only a single node.

**Spring Layout** - represents a visualization of a set of nodes. The Spring Layout assigns X/Y locations to each node and moves the image forward one step at a time. As a result, Spring Layout can take a considerable amount of time to execute.

**IHTSDO Workbench Graph Layout** - currently the same as the DAG Layout. Will be enhanced in future releases.

### Using the Conflicts View

If you have selected more than one path, the Conflicts toggle displays (or hides) a view of conflicts between the two paths, as shown in 

**Resolution**: The resolved (agreeing) components of the conflicting versions are listed in the upper left section of the Conflicts view.

**Versions**: The conflicting versions are identified in the two lower sections of the Conflicts view.

**Differences**: The boxes with colored lettering and borders indicate the conflict areas. To add conflicting concepts to the other version, click the “Add All” and “Implement” buttons.

**Long Form**: Click in this box to display more detailed definitions of each attribute displayed in the Conflict view.

**Status**: Click in this box to view the current status (current, etc.) of each attribute displayed in the Conflict view.

### Using the List View

The List View tab provides a way to conduct operations and execute batch changes on a list of child concepts related to the focus concept. When you click on the List tab, the List View window will display .
The List View window includes the business process plug-ins and toggle shown in .
Component View Toggle: Displays or hides component view of selected child concept.
Process Builder Plug-In: Displays or hides the process builder business process view.
Add Children to List Plug-In: Adds all of the children of the focus concept to a list, so operations can be performed on the list of child concepts. Added child concepts will be displayed in the “Destination Relationships” section of the Concepts Work Area.
Change Items to Selected Status Plug-In.
Reiterate Over Concepts Plug-In.
Mark as Current Plug-In.
Flag Concepts Plug-In.
Move Items to New Parent Plug-In.

Using the Search Function
The search function will typically be the first step in locating, changing and saving a concept and descriptions in the hierarchy. The search function area provides a way to search for a complete or partial concept name. The search reviews all of the hierarchies, or just the hierarchy specified in the Root field (Root rootof hierarchy) and then displays a list of every concept matching your search criteria. Using more advanced criteria, you can search within a specified portion of the hierarchy, within reference sets, or relationship type and destination.

Area Resizing Arrows

Click on these arrows to increase or decrease/hide the size of the search function area. You can also resize the search function area by hovering the cursor over the upper border of the area, then clicking and dragging the border to the desired position.

Query Field

To launch a search, use the query field to enter search criteria, i.e., the concept you want to locate in the hierarchy or hierarchies. Click on the binoculars icon to the right of the query field. You can also drag a concept from the taxonomy pane or from a field in the concept work pane.

Regex

Click on this option to search for regular expressions, i.e., very specific expressions that do not require specification of any variables.

Lucene

Click on this option to search for lucene expressions, which is the preferred search method. The Lucene search uses a rapid indexing method that produces results much faster than the Regex search option.

Lucene allows you to modify query terms, which provides a wide range of searching options.

Wildcard Searches

To perform a single character wildcard search, use the “?” symbol. A single character wildcard search looks for terms that match the single character replaced with the “?”.

To perform a multiple character wildcard search, use the “*” symbol. For example, to search for test, tests or tester, type “test*”.

You can also use the wildcard search in the middle of a term, such as “te*t”.

You cannot use a * or ? symbol as the first character of a search.

Fuzzy Searches

Fuzzy searches find terms that are similar, such as foam and roams.

To perform a fuzzy search, use the tilde ~ symbol at the end of a single word term. For example, to search for a term similar in spelling to “roam”, use the fuzzy search roam~.

Proximity Searches

A proximity search finds words within a specific distance away.

To perform a proximity search, use the tilde ~ symbol at the end of a phrase. For example, to search for aspirin and pressure within 10 words of each other in a document, type “pressure aspirin”~10

---

Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.
Boosting a Term

Lucene provides the relevance level of matching concepts based on the terms found. To boost a term, use the caret ^ symbol with a boost factor (a number) at the end of the term you are searching. The higher the boost factor, the more relevant the term.

For example, if you want the term “aspirin” to be more relevant, you can boost it using the ^ symbol along with the boost factor, such as “aspirin^4 pressure”.

This will make concepts with the term aspirin appear more relevant.

You can also boost phrase terms such as “aspirin pressure”^4 “hypertension treatment”
The default boost factor is 1.

Boolean Operators

Lucene supports Boolean terms AND, +, OR, NOT and -. Boolean operators must be in all caps. The OR operator is the default conjunction operator, so if there is no Boolean operator between two terms, OR is used.

**OR:** Links two terms and finds a matching document if either of the terms exist in a document. For example, to search for concepts that contain either aspirin pressure or just aspirin, type “aspirin pressure” aspirin or this variation: “aspirin pressure” OR aspirin

**AND:** Matches concepts where both terms exist anywhere in the text of a concept. To search for concepts containing aspirin pressure and hypertension treatment, type “aspirin pressure” AND “hypertension treatment”.

**+:** Requires that the term after the + symbol exist somewhere in a concept. To search for concepts that must contain aspirin and may contain pressure, type +aspirin pressure.

**NOT:** Excludes documents that contain the term after NOT. The symbol ! can be used in place of the word NOT.
To search for concepts that contain aspirin pressure but not hypertension treatment, type “aspirin pressure” NOT “hypertension treatment”.

Note: NOT cannot be used with just one term, such as NOT “aspirin pressure”.

The - or prohibit operator excludes concepts that contain the term after the - symbol. To search for concepts that contain aspirin pressure but not hypertension treatment, type “aspirin pressure” - “hypertension treatment”

Grouping

Lucene supports parentheses to group clauses to form sub-queries, which can be used to control the boolean logic for a query. To search for either aspirin or pressure and hypertension, type (aspirin OR pressure) AND hypertension.

**Status Kind - Advanced Search**

To further define a search using additional criteria, click on the Plus symbol on the left of the Query field. An additional search line, the “Status Kind” row, will appear, as shown in .

The drop-down list in the advanced search line is used to select the type of search criteria to use. Use the Exclude matches checkbox to retrieve concepts not matching the search criteria.

A number of advanced search lines can be “stacked” to filter results by multiple criteria. Click the plus sign button to add additional search lines. Click a search line’s “X” button to remove the field. Additional advanced searches include:

- **Status Kind** - filters a search by concept status (e.g., current, duplicate, retired, inactive, etc.). Only those concepts with a matching status type will be returned.
- **Is Child of** - filters a search by hierarchy branch. Only concepts under the selected parent concept will be returned.
- **Refset Member** - filters a search by returning only those concepts that are members of the specified refset.
Reliability /Kind - filters a search by relationship type (as entered in the type kind field) and the relationship destination (as entered in the restriction kind field).

**Binocular Icons**

Click on one of the binocular icons after entering your search criteria to launch the search. Both of the binocular icons in the search area perform the same function.

**Signpost Icon**

The signpost icon provides access to a scratch pad panel where developers can record task-specific instructions.

**List View Button**

Click the List View Button (notepad icon) to add results of the current search to the List View tab in the Concept Work Area.

**Hide Search Criteria Button**

Click the Hide Search Criteria Button to hide or reveal the current search criteria.

**Cancel Button**

Use the Cancel button to exit the current concept view without saving it.

**Commit Button**

Once you are satisfied with all of modifications and additions, use the Commit button to add the new or revised concept to the IHTSDO workspace.

**Score**

Using an algorithm, the Score ranks search results, based on how well the search phrase matches the resulting concepts.

**Text**

The text description of the search result.

**Root**

The root hierarchy - one of the hierarchies available in the Hierarchy View - to be searched. The field default is “Empty,” meaning all hierarchies will be searched. To specify a root, select a hierarchy from the Hierarchy View, then drag and drop or copy and paste the description into the Root field. The search will retrieve only the concepts available in the selected hierarchy.

---

**Using the Search Variables**

Searches can be executed in a number of ways, and can include variables to indicate inclusion or exclusion of words and phrases, as described below.

- Searching defaults to an “OR” search, not an “And/OR” search.
- To set up an “And/OR” search, insert a plus sign + in front of the search phrase. All search results will be required to include the search phrase.
- To exclude a word or phrase from the search, insert a minus sign - in front of the word or phrase you want to exclude. All search results will be required to exclude concepts containing the identified word or phrase.
- To search for descriptions that are similar to, but not exactly the same as, the search phrase, insert a tilde ~ at the end of the search phrase.
- All search results are considered to be an exact match unless the search phrase contains a tilde ~ at the end of the phrase.
- To search for descriptions related to a specific root, drag and drop the root description from the hierarchy view into the Root field.
Searching With Default Values

Searching for a concept without using variables produces an “OR” search, as shown.

Searching with the “Required” Variable

Searching for a concept that must include a specified word or phrase produces results such as those shown. Note that the “Score” column indicates how closely each concept matches the search criteria.

Searching with the “Exclude” Variable

Searching for a concept that must exclude a specified word or phrase produces results such as those shown. Again, note that the “Score” column indicates how closely each concept matches the search criteria.

Searching with Multiple Variables

Searches can include multiple variables, to indicate a combination of criteria, such as “similar to”, not including certain words or phrases, but must include other words or phrases, as shown. Again, note the “Score” column indicates how closely each concept matches the search criteria.
Saving and Retrieving Searches

You can save search strings that you use frequently by pressing the Save Search button. Enter a name for the search and click the Save button, as shown in . To retrieve a saved search, click the Retrieve Saved Search button. Select the desired search and click the Open button.

![Search - Save Button](image)

![Search - Retrieve Button](image)

Executing an Advanced Search

The search function can be used to execute advanced searches requiring more specific and complex identification of concepts. To illustrate this process, we will use ‘corneal abrasion’ as the initial search criteria.

Submitting a Change Set to the IHTSDO Workspace

All of the items related to a concept are called a Change Set. Changes to concepts, source and destination relationships are not logged in the IHTSDO workspace until they are committed into the Subversion version control system.

Using Reference Sets

A reference set (refset) is a set of concepts, descriptions, or relationships that are appropriate to a particular language, dialect, country, specialty, organization, user or context. A refset is a list of Universally Unique Identifiers (UUID). Each UUID refers to one component of the hierarchy, and makes it a member of the refset.

IHTSDO IDE includes two types of refsets: specifications and inclusion types.

Specification refsets are created by users, who explicitly define which concepts in the hierarchy are to be included in (or excluded from) the refset being defined.

Inclusion Types allow users to include or exclude individual concepts or whole branches of the concept hierarchy. These Specification refsets are processed by software algorithms in IHTSDO IDE to generate Member refsets, whose members contain all the concepts specified for inclusion in their corresponding Specification refset.

The following sections illustrate how to create a refset grouping containing concepts.

Note: IHTSDO IDE must be properly configured before you create any reference sets. IHTSDO IDE can be configured to display which concepts are members of a specific member refset. The diagonal paper clip icon next to a concept means that the concept is a member of a specified refset. A horizontal paper clip icon next to a concept indicates that one or more of its children is a member of the refset.

Creating a Member Refset Manually

While the automated function “Create refset and member/set pair” is the preferred method of creating reference sets, the following section describes how to create a member refset manually.

You must create a member refset before creating its corresponding specification refset. This process ensures that a destination exists for the IHTSDO IDE refset processing algorithm to populate when a specification refset is first specified.

Member Reference Set - Source Relationships
Creating a Specification Refset Manually

Once a member refset has been created, the next step is to create the Specification refset that IHTSDO IDE will use to populate it.

Specification Reference Set - Source Relationships

<table>
<thead>
<tr>
<th>Type</th>
<th>Hierarchy Location</th>
<th>Destination &amp; Hierarchy Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>is a</td>
<td>Terminology Auxiliary concept/relationship</td>
<td>refset/Refset Auxiliary Concept</td>
</tr>
<tr>
<td>refset type rel</td>
<td>Refset Auxiliary concept/ refset</td>
<td>concept extensions Refset Auxiliary concept/ refset type</td>
</tr>
</tbody>
</table>

Associating Concepts with a Specification Reference Set

To create a reference set, you must specify which concepts will be members - and which will not - in the specification refset.

The Refset Entries toggle, shown in displays or hides the links to reference sets for the selected concept.

Adding a Concept to a Specification Reference Set

To add the selected concept to an existing refset, click on the Add Concept Extension button, as shown in .

The default reset will appear in the refset column. Double-click this value to change to another refset. A drop-down list will display as shown in Select the desired refset.
Selecting the Concept Inclusion Type

When linking a concept to a refset, its inclusion type must be specified as either exclude individual, exclude lineage, individual include or lineage include.

When the concept is first added to a refset, an inclusion type will be assigned by default (as set in the Preferences Panel). To modify this value, double-click on it in the concept column. A list of values will display as shown in . Select the desired value. Or you can right-click on the entry and select Change concept to display the pop-up list.

Changing the Refset Association Status

The default refset status is normally set as current. (Other values are defined in the Refset preferences.) To modify this value, double-click on it in the status column. A list of values will display, as shown in . Select the desired value. Or you can right-click on the entry and select Change status to display the list.

Committing Changes

The refset associations entered are not final in the database until you click the Commit button at the bottom right of the IHTSDO IDE window. To clear the current entries (instead of committing them), click the Cancel button at the bottom right of the IHTSDO IDE window.

Displaying Specification Refset Concept Types in the Hierarchy List

IHTSDO IDE can be set to display icons in the hierarchy list next to those concepts that are members of one or more specification refsets. An icon is shown for each refset of which the concept is a member. There may be more than one icon per concept. The type of icon indicates the concept type associated with the membership in the refset, as shown in
The display of these icons must be enabled in the Preferences settings, under the View tab, on the refset sub-tab (under Refsets to show in taxonomy view). To add a refset, drag it from the hierarchy view pane onto the refset sub-tab, as shown in . To disable the display of icons for an associated refset, highlight the refset in the list and press the Delete key.

Displaying Refset Members in the Hierarchy List

The result of a specification refset is a member refset, i.e., the set of concepts specified by the specification refset. IHTSDO IDE can be set to display which concepts are members of a particular member refset.

A diagonal paper clip icon next to a concept indicates the concept is a member of the specified refset. A horizontal paper clip icon indicates that one or more of its children is a member of the refset. These icons can be displayed by dragging and dropping the desired member refset name to the refset sub-tab, as shown in .

Changing the Refset Default Values

The default parameters for the way IHTSDO IDE performs when linking a concept to a refset are specified in the Preferences panel under the Ref Set tab, under the sub-tabs Concept/concept/defaults. To change a default value, drag and drop the new value from the hierarchy panel to the default to be changed, as shown in .
Changing the Default Refset

When linking a concept to a refset, IHTSDO IDE initially specifies a default refset to link to (which can be changed afterwards if desired - see Adding a Concept to a Specification Reference Set.) To set a different default refset, drag and drop a refset name from the hierarchy view to the Default refset field of the tab shown in . The new value will overwrite the previous one.

Changing the Default Status

When a concept is linked to a refset, its initial status is normally set as current. To change this default value, drag and drop another status value from the hierarchy view to the Default status field of the tab shown in .

Changing the Concept Inclusion Type Default

When a concept is linked to a refset, IHTSDO IDE initially specifies an inclusion type (which can be modified later as described in Adding a Concept to a Specification Reference Set). To choose a different default inclusion type, drag and drop your selection from the hierarchy view to the Default concept field of the tab shown in . The new value will overwrite the previous one.

Using Refset Type Tab

The refset types tab, as shown in  is used to identify the refsets that will be available for selection when adding a concept to a refset. To add a refset to the list, click and drag refset values from the terminology pane (under Refset Auxiliary concept/refset) onto the tab. To delete a refset from the list, highlight it and press the Delete button.
**Status Types**

The status types tab, as shown in sets the status values that will be available when adding a concept to a refset. To add a status to the list, click and drag status type values from the terminology pane (under Terminology Auxiliary Concept/status) onto the tab. To delete a status type from the list, highlight it and press the Delete button.

**Concept Types Tab**

The concept types tab, as shown in sets the concept values that will be available when adding a concept to a refset. To add a concept to the list for a concept refset, click and drag concept type values from the terminology pane (under Refset Auxiliary Concept/refset) onto the tab. To delete a refset from the list, highlight it and press the Delete button.
Setting Refset Preferences

To configure IHTSDO IDE to display refset information in the taxonomy view pane:

Under the Refset tab, on the Taxonomy sub-tab, ensure show viewer images in taxonomy view and show refset info in taxonomy view are both checked, as shown in .

Next, on the Component Panel tab, ensure that REFSETS is checked, as shown in .
Inclusion Types

To rule in or out concepts in the SNOMED hierarchy for membership in a refset, you must use the appropriate Inclusion Type. An inclusion type determines whether a selected concept will be explicitly included or explicitly excluded from a reference set. It can also specify whether the children of a concept will (together with the parent) be included or excluded from the reference set.

Inclusion types are used to define which members of the hierarchy will be included in a given refset. Rather than manually flagging each concept to be included, inclusion types define branches of the hierarchy to include. If some sub-branches or individual concepts are to be excluded from the refset, they may be flagged with “exception” inclusion types.

There are four inclusion types and each has a symbol associated with it, as shown in . These symbols appear in the taxonomy view pane, next to concepts that have a refset membership definition associated with them, as shown in .
**Individual Include**

Individual Include means that the selected concept should be included in the member refset. Its child concepts, if any, are not included.

**Lineage Include**

Lineage Include means that the selected concept and all of its children (and sub-children) will be included in the member refset. However, only concepts whose status is either current or pending move will be included. If a concept has any other status (e.g., retired or inactive), it will not be included in the member refset, nor will any of its children or sub-children, regardless of their status.

Further, if a parent concept has a status of current or pending move, but one or more of its children or sub-children does not, the parent will be included but the children or sub-children will be excluded from the member refset, as will their children and sub-children.

**Exclude Individual**

Exclude Individual means to exclude a concept from the member refset. If the concept has any child concepts, they will still be included.

**Exclude Lineage**

Exclude Lineage means that this concept and all of its children (and sub-children) will be excluded from the member refset.

**Inclusion Types and Multiple Inheritance**

A concept may have multiple inheritances (i.e., it may appear in more than one place in the SNOMED hierarchy). Such concepts are indicated by a green up-arrow symbol in the taxonomy view pane. When specifying a refset using inclusion types, it is possible to include one instance of such a concept (either individually or by lineage) and exclude another, leading to a conflict. *(The same concept cannot be simultaneously both included and excluded from membership in a refset.)* The algorithm that generates the member refset will flag this as an error. To resolve such conflicts, you must ensure that each instance of a concept with multiple inheritance is either consistently included or excluded.

The hierarchy fragment shown in displays an example of this type of conflict. The concept Alcoholic Beverage has multiple inheritance. It appears as a sub-child of the concept Dietary substance and also as a sub-child of the concept Drinks.
Substance of abuse. Dietary substance has the lineage include type applied to it, and thus its sub-child Alcoholic beverage will be excluded from the refset. The concept Alcoholic beverage cannot be both included and excluded; the two options are mutually exclusive. Unless this situation is resolved, a member refset cannot be generated.

**Conclusion**

This manual is designed for experienced or “power” users of the hierarchies and concept building tools. Scripts that control the user environment and provide guidance to available options may be more appropriate for other users.
Setting Preferences

The behaviour of most IHTSDO IDE parameters is set in the Preferences panel. To display the Preferences tabs, click on the Display Preferences button. The sub-tabs displayed depend upon which parent tab is selected.
The Preferences Tabs

Preference values are set for the current configuration. Different user profiles can be saved with different preference settings and loaded when the workbench is started.

Values in the preferences tabs fields are populated by dragging a value from another area of the screen (usually the taxonomy view pane) onto the field to be populated, as shown in . To delete a value from a tab, highlight it and press the Delete button on the keyboard.

Click again on the Preferences button to hide the Preferences tabs.

View Tab

The View tab is used to control the display of items in the taxonomy view pane, how fields are displayed in the various sections of the work area.

Descriptions

Short label preference order
Specifies how to display the concept name in those areas of the screen that use short labels (e.g. the taxonomy pane), based on the concept description type. The first concept description whose description type matches one of description types in the list (in descending order) will be the one displayed for that concept.

Long label preference order
Specifies how to display the concept name in those areas of the screen that use long labels (e.g. the concept work area), based on the concept description type (e.g. fully specified name, preferred term). The first concept description whose description type matches one of description types in the list (in descending order) will be the one displayed for that concept.

Tree preference order
Specifies how to display the concept name in the taxonomy view pane, based on the concept description type (e.g. fully specified name, preferred term). The first concept description whose description type matches one of description types in the list (in descending order) will be the one displayed for that concept.

Table preference order

Hierarchy Roots
Defines the highest level (root) concepts in the taxonomy. All other concepts will be children or sub-children of these concepts.
Taxonomy
Display toggles
Use inferred rels in taxonomy view
Allow variable height taxonomy view
Show viewer images in taxonomy view
Show refset info in taxonomy view
Sort taxonomy using refset

Parent relationships
Defines the relationships used to associated parent concepts with child concepts.

Child relationships
Defines the relationships used to associated child concepts with parent concepts.

Stated view characteristic types
Inferred view characteristic types

Taxonomy sort
If the Sort taxonomy using refset is selected on the Taxonomy tab (see previous section), defines the refsets that will order the taxonomy display.

Status
Defines which concepts will be displayed in the taxonomy view pane based on their value in their status field.

Refsets to show in taxonomy view
Specifies the refsets for which concepts in the hierarchy display will show adjacent inclusion or exclusion symbols.

Edit Tab

Defaults

Default status
Specifies the initial value of the status field when a new concept is added.

Default image type
Specifies the initial value of the status field in the Image section of the work area when associating an image with a concept.

Default description type
Specifies the initial value of the description that will appear in the type field of the Descriptions section of the work area when a new concept is added.

Default relationship type
Specifies the initial value in the type field of the Source relationships and Destination relationships sections of the work area when a new concept is added.

Default relationship characteristic
Specifies the initial value in the char (characteristic) field of the Source relationships and Destination relationships sections of the work area when a new concept is added.

Default relationship refinability
Specifies the initial value in the refinability field of the Source relationships and Destination relationships sections of the work area when a new concept is added.
Relationship Types (values for popup)
Defines the values available for the relationship type of a concept. These values specify the choices displayed in the type field popup selection boxes in the Source relationships and Destination relationships sections of the work area.

Relationship Refinability (values for popup)
Defines the values available for the refinability of a concept. These values specify the choices displayed in the refinability field popup selection boxes in the Source relationships and Destination relationships sections of the work area.

Relationship Characteristics
Defines the values available for the characteristics of a concept. These values specify the choices displayed in the char (characteristic) field popup selection boxes in the Source relationships and Destination relationships sections of the work area.

Description Types
Defines the values available for the description type of a concept. These values specify the choices displayed in the type field popup selection box in the Descriptions section of the work area.

Image Types
Defines the values available for a concept’s image type. These values specify the choices displayed in the type field popup selection box in the Images section of the work area.

Status Values
Defines the values available for the status value of a concept. These values specify the choices displayed in the status field popup selection boxes in the Concept attributes, Refset, Descriptions, Source relationships, Destination relationships, and Images sections of the work area.

Path Tab
Terminology Auxiliary -
Target terminology independent concepts used by the IHTSDO Workbench Editor
.SNOMED Core -SNOMED CT Terminology concepts, descriptions and relationships distributed by IHTSDO.

New Path Tab
Allows for the creation of a new editing path
- Description for new path-- TBD
- Parent for new path-- OK?
- Create button

Reference Set Tab
This tab applies to concept reference sets as described in Reference Sets. For details of reference set preference settings, see section Setting Refset Preferences.
Component Panel Tab

The checkboxes on the Component panel tab determine which display toggle buttons are shown or hidden. Hiding a button ensures that the corresponding portion of the work area is either always displayed or always hidden (i.e., cannot be toggled on or off).
### Adding and Modifying Concepts

<table>
<thead>
<tr>
<th>Concepts</th>
<th>A concept is a clinical meaning identified by a unique name. Concepts are formally defined in terms of their relationships with other concepts. These “logical definitions” give explicit meaning which a computer can process and query on. Every concept also has a set of terms that name the concept in a human-readable way.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationships</td>
<td></td>
</tr>
<tr>
<td>Submitting a Change Set to the IHTSDO Workspace</td>
<td></td>
</tr>
</tbody>
</table>
**Concepts**

**Searching for a Concept**

The Search feature is the most efficient way to find a particular concept in the hierarchy. When you launch a basic search, the IHTSDO IDE looks through every terminology list in the database and displays a list of all concepts matching the search criteria. Using more advanced search criteria, you can search within a specified portion of the hierarchy, within reference sets, or within relationship type and destination.

**Basic Search**

To launch a search, type the search criteria in the query field in the Search function area of the window. Click on the binoculars icon to the right of the query field. Alternatively, you can drag a concept from the taxonomy pane or from a field in the concept work pane. For example, a search for all concepts containing “ocular refraction” would begin as shown in . A “searching for concept” message will display while the search proceeds. (Note that when more than one word is entered, matches for each word are returned. To search for a phrase, enclose it in double quotes.)

When the search is finished, the text fields in the Search area will display a list of concepts matching the search criteria, as shown in . The list can be quite long, depending on the search term and how the criteria narrow the search. To view more of the list, click on the resizing arrows on the top left of the Search area to expand the list view.

To select a concept from the search results, click on it. The text on the Search tab at the top of the window will change to the name of the selected concept, as shown in . To see the details of this concept, click on this tab.

The selected concept, or “focus concept”, will display in the first line of the Descriptions section of the work area, as shown in . Any additional relationships that this concept has are displayed on succeeding lines.
Advanced Search

In addition to the basic search capability described in the preceding section, it is possible to search using additional criteria. To initiate an advanced search, click the green “plus sign” button to the left of the query field. An additional search line will appear below the first one, as shown in .

The dropdown box in the advanced search line is used to select the type of search criteria to use. Use the Exclude matches checkbox to return concepts not matching the search criteria.

A number of advanced search lines can be “stacked” to filter results by multiple criteria; click the “plus sign” button to add additional search lines. Click a search line’s “X” button to remove the field.

Status Kind—The status kind search filters a search by concept status (e.g. current, duplicate, retired). Only concepts with a matching status type will be returned.

Is Child of—The is child of search filters a search by hierarchy branch. Only concepts under the selected parent concept will be returned.

Refset Member—The refset member search filters a search by only returning those concepts that are members of the specified refset

Relationship/Kind—The rel kind search filters a search by relationship type (as entered in the type kind field) and the relationship destination (as entered in the restriction kind field).

Saving Searches

You can save search strings that you use frequently. To save a search string, press the Save Search button. Enter a name for the search and click the Save button. To retrieve a saved search, click the Retrieve Saved Search button. Select the desired search and click the Open button.

Concept Description Fields

Text The concept description.

Type Represents the type of the description, e.g., Fully Specified Name, Preferred Term, or Synonym.

Case Indicates whether or not the capitalisation of the term is significant.

Lang Language used to store and display the description, e.g., en = English.

Status Indicates whether the concept description is current (e.g. current, limited, pending move) or inactive (e.g. retired, duplicate, outdated).
Version The version of the terminology in which the description was created. If the description has not yet been added to a change set, the Version field will display “uncommitted.” (Only appears when history records are displayed.)

Path The name of the terminology version in which the description was created. (Only appears when history records are displayed.)

**Adding a New Description**

To add a new concept, click on the “add new description” icon in the Descriptions section of the work area. A new line will be added to the list, with the default description text “New Description”, as shown in .

Note that new lines that have not yet been committed to the database are identified by a green vertical bar at the left-hand end of the line. This visual identifier is used throughout the IHTSDO IDE to denote uncommitted additions. The green bar will remain until you press either the commit or cancel button.

To enter the name of the new concept, double-click on the text field in the new line. The text “New Description” will become editable (a cursor will appear). Replace it with the name of the new concept.

Double-click on the type field and select one of the available options in the drop-down menu.

Double-click on the case field and select True or False from the drop-down menu.

The lang (language) field will default to English (en), unless the IHTSDO Workbench Viewer has been configured for a different environment.

Double-click on the status field and select one of the available options in the drop-down menu:

- Current
- Retired
- Duplicate
- Outdated
- Ambiguous
- Erroneous
- Limited
- Inappropriate
- Concept retired
- Moved elsewhere
- Pending move

The version and path fields are only displayed when the “Display history records” button is depressed. These information fields are automatically updated by IHTSDO IDE.

To save the new concept, click the commit button.

**Modifying an Existing Concept**

To modify an existing concept, right-click on the concept description to be modified in the Descriptions section of the work area. The options Change, Change Type, and Change Status are displayed. Select the property to modify and select a new value from the corresponding popup selection box, as shown in .

To save the modified concept, click the commit button.
Concept Attributes

Most concepts are defined, and their defined attribute is set as true. However, there are abstract or theoretical concepts that are not defined, for example, the cornerstone concept at the top of a hierarchy. For such concepts, the value of the defined attribute should be set as false.

The status of a concept indicates whether it is in active use and, if not, indicates the reason for withdrawal from use.

To change the value of the defined attribute of a concept, right-click on it in the Concept attributes section of the work area and select Change Defined.

To change the status of a concept, right-click on it and select Change Status. The choices that appear are set in Preferences.

Adding a New Concept

To create a new concept, first select a parent concept for it in the hierarchy viewer. Then, click the “create new concept” button. Enter values for the concept’s attributes and descriptions as described in the sections above.

Relationships

Relationships link concepts to one another. “is a” relationships link concepts in a “parent-child” way (e.g. duck is a bird; bird is a animal). A concept can have more than one “is a” relationship to other concepts (e.g. bird is a animal, bird is a vertebrate).

Attribute relationships form a class of similar relationships that relate two concepts and establish the type of relationship between them. These types of relationships are defined by the designer of the hierarchy/database. For example, the attribute relationships in a pharmacology database could include Has dose form and Has active ingredient.

A relationship exists between two concepts. That relationship may appear as either a source relationship or a destination relationship -- depending on which of the two concepts’ details you are viewing. Relationships with concepts higher up a hierarchy than the concept are source relationships. The IHTSDO IDE displays both source and destination relationships for a concept.
**Source Relationship Fields**

- **Type**: Represents the type of relationship between two related concepts (e.g. is a, Has active ingredient, Causative agent).
- **Destination**: The concept to which this concept is connected in the terminology hierarchy.
- **Char**: Characteristic type of relationship, e.g. defining, qualifying, historical, additional.
- **Refinability**: Identifies the refinability type of the relationship. May be: Not Refinable, Optional or Mandatory.
- **Group**: SNOMED role group.
- **Status**: Indicates whether the relationship is current (e.g. current, limited, pending move) or inactive (e.g. retired, duplicate, outdated).

**Destination Relationship Fields**

- **Origin**: The concept to which this concept is connected in the terminology hierarchy.
- **Type**: Represents the type of relationship between two related concepts (e.g. is a, Has active ingredient, Causative agent).
- **Char**: Characteristic type of relationship, e.g. defining, qualifying, historical, additional.
- **Refinability**: Identifies the refinability type of the relationship. May be either: Not Refinable, Optional or Mandatory.
- **Status**: Indicates whether the relationship is current (e.g. current, limited, pending move) or inactive (e.g. retired, duplicate, outdated).

**Adding a Source Relationship**

To create a new source relationship, click on the “add” icon in the Source relationships section of the work area. A new line will be added to the list, with the default relationship type “is-a rel (terminology constant)”, as shown in .

Double-click on the type field and select one of the options in the drop-down menu. Alternatively, drag a valid value from the taxonomy view pane into the type field of the relationship.

Select values for the char (characteristics), refinability, and status attributes of the relationship in the same way.

**Modifying a Source Relationship**

To modify an existing source relationship, begin by right-clicking anywhere on the line containing it. A drop-down showing the attributes available for change is shown, as shown in .

**Destination Relationships**

It is possible to edit destination relationships without having to browse directly to the source concept. The process is the same as for editing a source relationship.
Submitting a Change Set to the IHTSDO Workspace

All of the items related to a concept are called a Change Set. Changes to concepts, source and destination relationships are not logged in the IHTSDO workspace until they are committed into the Subversion version control system.

When you are satisfied with the changes to the focus concept, click on the Subversion icon. The Change Sets panel will display, as shown in.

To commit changes set to the IHTSDO workspace, click on the commit button. The lower part of the panel will display a message, indicating the system time and “Starting Commit.” When the commit is finished, click on the update button to retrieve updates committed into Subversion by other users.

Change Set panel buttons:

status Lists all files that have local modifications.
commit Updates working copy; only prints information about files updated.
update Brings changes from the repository into your working copy.
cleanup Runs any unexecuted log files and removes working copy locks.
get Performs a Subversion checkout.
 purge Removes files selected for deletion.
clear log Clears all log text from the Change Sets Panel window.
Data Checks and Search Filters

Searching for a Concept
DataChecks

One key aspect of the quality assurance process for terminology development is undertaking a set of checks on new concepts to ensure they respect the editorial rules and data models of the particular terminology. Sometimes these rules are documented explicitly. Sometimes there are some implicit rules that need to be observed. The IHTSDO Toolkit provides a number of tasks and functions that support data checking.
Searching for a Concept

The Search feature is the most efficient way to find a particular concept in the hierarchy. When you launch a basic search, the IHTSDO IDE looks through every terminology list in the database and displays a list of all concepts matching the search criteria. Using more advanced search criteria, you can search within a specified portion of the hierarchy, within reference sets, or within relationship type and destination.

Basic Search

To launch a search, type the search criteria in the query field in the Search function area of the window. Click on the binoculars icon to the right of the query field. Alternatively, you can drag a concept from the taxonomy pane or from a field in the concept work pane. For example, a search for all concepts containing “ocular refraction” would begin as shown in . A “searching for concept” message will display while the search proceeds. (Note that when more than one word is entered, matches for each word are returned. To search for a phrase, enclose it in double quotes.)

When the search is finished, the text fields in the Search area will display a list of concepts matching the search criteria, as shown in . The list can be quite long, depending on the search term and how the criteria narrow the search. To view more of the list, click on the resizing arrows on the top left of the Search area to expand the list view.

To select a concept from the search results, click on it. The text on the Search tab at the top of the window will change to the name of the selected concept, as shown in . To see the details of this concept, click on this tab.

The selected concept, or “focus concept”, will display in the first line of the Descriptions section of the work area, as shown in . Any additional relationships that this concept has are displayed on succeeding lines.
Advanced Search

In addition to the basic search capability described in the preceding section, it is possible to search using additional criteria. To initiate an advanced search, click the green “+” button to the left of the query field. An additional search line will appear below the first one, as shown in.

The dropdown box in the advanced search line is used to select the type of search criteria to use. Use the Exclude matches checkbox to return concepts not matching the search criteria.

A number of advanced search lines can be “stacked” to filter results by multiple criteria; click the “+” button to add additional search lines. Click a search line’s “X” button to remove the field.

**Status Kind:** The status kind search filters a search by concept status (e.g. current, duplicate, retired). Only concepts with a matching status type will be returned.

**Is Child of:** The is child of search filters a search by hierarchy branch. Only concepts under the selected parent concept will be returned.

**Refset Member:** The refset member search filters a search by only returning those concepts that are members of the specified refset.

**Relationship/Kind:** The rel kind search filters a search by relationship type (as entered in the type kind field) and the relationship destination (as entered in the restriction kind field).

Saving Searches

You can save search strings that you use frequently. To save a search string, press the Save Search button. Enter a name for the search and click the Save button. To retrieve a saved search, click the Retrieve Saved Search button. Select the desired search and click the Open button.

DataChecks

All of the items related to a concept are called a Change Set. Changes to concepts, source and destination relationships are not logged in the IHTSDO workspace until they are committed into the Subversion version control system. Before committing a local editing change to a new change set, or before committing the change set to the
subversion repository, the integrity of the proposed changes should be verified against editorial rules and policies and against the relevant terminology concept model. The IHTSDO Toolkit already provides a number of data check tasks to assist this process, but further terminology- and realm-specific datachecks may be required in addition.
A refset is a set of concepts, descriptions, or relationships that are appropriate to a particular language, dialect, country, specialty, organization, user or context. In its simplest form, a refset is a list of Universally Unique Identifiers (UUIDs). Each UUID refers to one component of the hierarchy, and makes it a member of the refset (a refset member).

In IHTSDO IDE, there are two types of refsets: Specification refsets are created by users, who explicitly define which concepts in the hierarchy are to be included in (or excluded from) the refset being defined. Using Inclusion Types, users may include or exclude individual concepts or whole branches of the concept hierarchy. (See Inclusion Types for a detailed explanation of how inclusion types work.) These Specification refsets are processed by software algorithms in IHTSDO Workbench to generate Member refsets, whose members contain all the concepts specified for inclusion in their corresponding Specification refset.

The following sections illustrate how to create a refset grouping containing concepts.

Before creating any reference sets, IHTSDO IDE must be properly configured. See Setting Preferences.
Creating a Member Refset Manually

Create a member refset first, before creating its corresponding specification refset. This is to ensure that a destination exists for the IHTSDO Workbench refset processing algorithm to populate when a specification refset is first specified.

<table>
<thead>
<tr>
<th>Type</th>
<th>Hierarchy Location</th>
<th>Destination</th>
<th>Hierarchy Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>is a</td>
<td>Terminology Auxiliary concept /relationship</td>
<td>refset</td>
<td>Refset Auxiliary Concept</td>
</tr>
<tr>
<td>refset type rel</td>
<td>Refset Auxiliary concept /refset relationship</td>
<td>concept extensions</td>
<td>Refset Auxiliary concept /refset type</td>
</tr>
</tbody>
</table>

Creating a Specification Refset Manually

Once a Member refset has been created, the next step is to create the Specification refset that IHTSDO Workbench will use to populate it.

<table>
<thead>
<tr>
<th>Type</th>
<th>Hierarchy Location</th>
<th>Destination</th>
<th>Hierarchy Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>is a</td>
<td>Terminology Auxiliary concept /relationship</td>
<td>refset</td>
<td>Refset Auxiliary Concept</td>
</tr>
<tr>
<td>refset type rel</td>
<td>Refset Auxiliary concept /refset relationship</td>
<td>concept extensions</td>
<td>Refset Auxiliary concept /refset type</td>
</tr>
<tr>
<td>refset purpose rel</td>
<td>Refset Auxiliary concept /refset relationship</td>
<td>(depends on purpose, e.g. inclusion specification)</td>
<td>Refset Auxiliary concept /refset purpose</td>
</tr>
<tr>
<td>generates</td>
<td>Refset Auxiliary concept /refset relationship</td>
<td>(name of the member refset)</td>
<td>Refset Auxiliary concept /refset</td>
</tr>
</tbody>
</table>
Associating Concepts with a Specification Reference Set

To create a reference set, you must specify which concepts will be members (and which will not) in the specification refset.

**Refset Entries**

The Refset Entries toggle button displays or hides the links to reference sets for the selected concept.

**Adding a Concept to a Specification Reference Set**

To add the selected concept to an existing refset, click on the Add Concept Extension button. The default refset will appear in the refset column. To change this value to another refset, double-click it. A dropdown box will appear with the names of all existing refsets; select the desired one. (To select the default refset, see section below.

**Selecting the Concept Inclusion Type**

When linking a concept to a refset, its inclusion type must be specified as either exclude individual, exclude lineage, individual include, or lineage include. (See section for a detailed description of these inclusion types.)

When the concept is first added to a reference set, an inclusion type will be assigned by default (as set in the Preferences panel). To modify this value, double-click on it in the concept column. A list of values will pop-up; select the desired value. Alternatively, right click on the entry and select Change concept to display the pop-up list.

**Changing the Refset Association Status**

The default refset status is normally set as current. (Other values are defined in the Refset preferences.) To modify this value, double-click on it in the status column. A list of values will pop-up; select the desired value. Alternatively, right-click on the entry and select Change status to display the pop-up list.
Committing Changes

The refset associations entered are provisional. No changes are made to the database until you click the commit button (at the bottom right of the window). If you do not wish to apply these entries, click the cancel button (next to the commit button) to clear the provisional refset associations from the display. They will not be applied to the database.

Displaying Specification Refset Concept Types in the Hierarchy List

IHTSDO Workbench can be set to display icons in the hierarchy list next to those concepts that are members of one or more specification refsets. An icon is shown for each refset of which the concept is a member; therefore there may be more than one icon per concept. The type of icon indicates the concept type associated with the membership in the refset, as shown in the table below:

<table>
<thead>
<tr>
<th>Type of Icon</th>
<th>Concept Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>exclude individual</td>
<td>individual include</td>
</tr>
<tr>
<td>exclude lineage</td>
<td>lineage include</td>
</tr>
</tbody>
</table>

The display of these icons must be enabled in the Preferences settings, under the View tab, on the refset sub-tab (under Refsets to show in taxonomy view). To add a refset, drag it from the hierarchy view pane onto the refset sub-tab. To disable the display of the icons for an associated refset, highlight the refset in the list and press the Delete key.
Displaying Refset Members in the Hierarchy List

The result of a specification refset is a member refset, i.e., the set of concepts specified by the specification refset. IHTSDO Workbench can be set to display which concepts are members of a particular member refset.

A diagonal paper clip icon next to a concept means that the concept is a member of the specified refset. A horizontal paper clip icon next to a concept indicates that one or more of its children is a member of the refset.

The display of these icons is enabled by dragging the desired member refset name onto the refset sub-tab, as shown in the previous section.

Changing the Refset Default Values

The default parameters for the way IHTSDO IDE behaves when linking a concept to a refset are specified in the Preferences panel under the Ref Set tab, under the sub-tabs Concept\concept\defaults. To change a default value, drag the new value desired from the hierarchy panel onto the default to be changed.

Default Refset

When linking a concept to a refset, IHTSDO IDE initially specifies a default refset to link to (which can be changed afterwards if desired; see section ). To set a different default refset, drag a different refset name from the hierarchy view pane onto the Default refset field of the tab. The new value will overwrite the previous one.

Default Status

When a concept is linked to a refset, its initial status is normally set as current. To change this default value, drag another status value from the hierarchy view pane onto the Default status field of the tab.

Concept Inclusion Type Default

When a concept is linked to a refset, IHTSDO IDE initially specifies an inclusion type (which can be modified afterwards; see section ). To choose a different default inclusion type, drag one from the hierarchy view pane onto the Default concept field of the tab. The new value will overwrite the previous one.
Refset Types Tab
The refset types tab is used to set which refsets will be available for selection when adding a concept to a refset.
To add a refset to the list, click and drag refset values from the terminology pane (under Refset Auxiliary concept \refset) onto the tab. To delete a refset from the list, highlight it and press the Delete button.

Status Types Tab
The status types tab sets the status values that will be available when adding a concept to a refset.
To add a status to the list, click and drag status type values from the terminology pane (under Terminology Auxiliary concept\status) onto the tab. To delete a status type from the list, highlight it and press the Delete button.

Concept Types Tab
The concept types tab sets the concept values that will be available when adding a concept to a refset.
To add a concept to the list for a concept refset, click and drag concept type values from the terminology pane (under Refset Auxiliary Concept\refset) onto the tab. To delete a refset from the list, highlight it and press the Delete button.
Refset Preferences

To set up IHTSDO IDE to display refset information in the taxonomy view pane:

Under the Ref Set tab, on the Taxonomy sub-tab, ensure show viewer images in taxonomy view and show refset info in taxonomy view are both checked.

On the Component Panel tab, ensure that REFSETS is checked.
In order to rule in or rule out concepts in the SNOMED hierarchy for membership in a reference set (refset), you must use the appropriate Inclusion Type.

An inclusion type determines whether a selected concept will be explicitly included or explicitly excluded from a reference set. It can also specify whether the children of a concept will (together with the parent) be included or excluded from the reference set.

Inclusion types are used to define which members of the hierarchy will be included in a given refset. Rather than manually flagging each concept to be included, inclusion types define branches of the hierarchy to include. If some sub-branches or individual concepts are to be excluded from the refset, they may be flagged with “exception” inclusion types.

There are four inclusion types; each has a symbol associated with it. These symbols appear in the taxonomy view pane, next to concepts that have a refset membership definition associated with them, as shown in .
Individual Include

Individual Include means that the selected concept should be included in the member refset. Its child concepts, if any, are not included.

Lineage Include

Lineage Include means that the selected concept and all of its children (and sub-children) will be included in the member refset. However, only concepts whose status is either current or pending move will be included. If a concept has any other status (e.g. retired or inactive), it will not be included in the member refset, nor will any of its children or sub-children, regardless of their status.

Furthermore, if a parent concept has a status of current or pending move but one or more of its children or sub-children does not, the parent will be included but those children or sub-children will be excluded from the member refset, as will their children and sub-children.

Exclude Individual

Exclude Individual means to exclude a concept from the member refset. If the concept has any child concepts, they will still be included.

Exclude Lineage

Exclude Lineage means that this concept and all of its children (and sub-children) will be excluded from the member refset.

Inclusion Types and Multiple Inheritance

A concept may have multiple inheritance (i.e., it may appear in more than one place in the SNOMED hierarchy).

Such concepts are indicated by a (green up-arrow) symbol next to them in the taxonomy view pane. When specifying a refset using inclusion types, it is possible to include one instance of such a concept (either individually or
by lineage) and exclude another, leading to a conflict. (The same concept cannot be simultaneously both included and excluded from membership in a refset.) The algorithm that generates the member refset will flag this as an error. To resolve such conflicts, you must ensure that each instance of a concept with multiple inheritance is either consistently included or excluded.

The hierarchy fragment shown in Figure 35 shows an example of this type of conflict. The concept Alcoholic beverage has multiple inheritance. It appears as a sub-child of the concept Dietary substance and also as a sub-child of the concept Substance of abuse. Dietary substance has the lineage include inclusion type applied to it, and thus its sub-child Alcoholic beverage will be included in the member refset. But Substance of abuse has the exclude lineage inclusion type applied, meaning that its sub-child Alcoholic beverage will be excluded from the refset. The concept Alcoholic beverage cannot be both included and excluded; the two options are mutually exclusive. Unless this situation is resolved, a member refset cannot be generated.
The following section describes each task from the standard task library that is supplied with the Distributed Workflow Architecture (DWFA) framework. The tasks appear below sorted by task purpose.

A task represents the instructions and data requirements for a single piece of work to be undertaken by a person or software process. The tasks available within the IHTSDO Workbench IDE determine how well the editor can meet the workflow requirements for terminology development.
### Process control: flow tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Availability in IDE</th>
<th>Implementation Package</th>
<th>Implementation Class</th>
<th>Task Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aggregator</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability in IDE:</td>
<td>Task list navigator, under:</td>
<td>flow tasks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation Package:</td>
<td>org.dwfa.bpa.tasks.util</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation Class:</td>
<td>Aggregator.java</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Properties:</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Change instance Id</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Generates and changes to a new Id for the current process instance. The Id is a Universally Unique ID (UUID), randomly generated at execution time. The process instance can be recognised from others of the same type or name using this ID.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability in IDE:</td>
<td>Task list navigator, under:</td>
<td>flow tasks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation Package:</td>
<td>org.dwfa.bpa.tasks.util</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation Class:</td>
<td>ChangeProcessInstanceId.java</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Properties:</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Abort workflow transaction</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>aborts the current transaction in the currently running process.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability in IDE:</td>
<td>Task list navigator, under:</td>
<td>flow tasks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation Package:</td>
<td>org.dwfa.bpa.tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation Class:</td>
<td>AbortWorkflowTransaction.java</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Properties:</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Random branch</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>Enables a process to randomly branch from up to 100%.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability in IDE:</td>
<td>Task list navigator, under:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation Package:</td>
<td>flow tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation Class:</td>
<td>RandomBranch.java</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Properties:</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Process complete

<table>
<thead>
<tr>
<th>Description:</th>
<th>Sets the condition status of the current business process to complete.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability in IDE:</td>
<td>Task list navigator, under: flow tasks</td>
</tr>
<tr>
<td>Implementation Package:</td>
<td>org.dwfa.bpa.tasks.util</td>
</tr>
<tr>
<td>Implementation Class:</td>
<td>Complete.java</td>
</tr>
<tr>
<td>Task Properties:</td>
<td>none</td>
</tr>
</tbody>
</table>

### Stop

<table>
<thead>
<tr>
<th>Description:</th>
<th>Sets the condition status of the current process instance to STOP. There may be multiple stop points in a Business Process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability in IDE:</td>
<td>Task list navigator, under: flow tasks</td>
</tr>
<tr>
<td>Implementation Package:</td>
<td>org.dwfa.bpa.tasks.util</td>
</tr>
<tr>
<td>Implementation Class:</td>
<td>StopProcess.java</td>
</tr>
<tr>
<td>Task Properties:</td>
<td>None</td>
</tr>
</tbody>
</table>

**Task: Stop**

- **Description:** Sets the condition status of the current process instance to STOP. There may be multiple stop points in a Business Process.
- **Availability in IDE:** Task list navigator, under: flow tasks
- **Implementation Package:** org.dwfa.bpa.tasks.util
- **Implementation Class:** StopProcess.java
- **Task Properties:** None

### Process control: Start tasks
<table>
<thead>
<tr>
<th><strong>Launch Process from Internal Task</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Availability in IDE:</strong></td>
</tr>
<tr>
<td><strong>Implementation Package:</strong></td>
</tr>
<tr>
<td><strong>Implementation Class:</strong></td>
</tr>
<tr>
<td><strong>Task Properties:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Launch Process From Container</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Availability in IDE:</strong></td>
</tr>
<tr>
<td><strong>Implementation Package:</strong></td>
</tr>
<tr>
<td><strong>Implementation Class:</strong></td>
</tr>
<tr>
<td><strong>Task Properties:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Launch Process From URL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Availability in IDE:</strong></td>
</tr>
<tr>
<td><strong>Implementation Package:</strong></td>
</tr>
<tr>
<td><strong>Implementation Class:</strong></td>
</tr>
<tr>
<td><strong>Task Properties:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Load, Set, Launch Process From URL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
</tbody>
</table>
## Load, Set, Launch Process From URL

<table>
<thead>
<tr>
<th>Availability in IDE:</th>
<th>Task list navigator, under: processes # start tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation Package:</td>
<td>org.dwfa.bpa.tasks.process</td>
</tr>
<tr>
<td>Implementation Class:</td>
<td>LoadSetLaunchProcessFromURL.java</td>
</tr>
<tr>
<td>Task Properties:</td>
<td>processURLString: A URL from which a process is loaded.</td>
</tr>
</tbody>
</table>

## Load, Set, Launch process from Attachment

<table>
<thead>
<tr>
<th>Description:</th>
<th>This task loads a process from an attachment. It sets all the external properties of the loaded process to the value of the properties of the same name and type from within the calling process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability in IDE:</td>
<td>Task list navigator, under: processes # start tasks</td>
</tr>
<tr>
<td>Implementation Package:</td>
<td></td>
</tr>
<tr>
<td>Implementation Class:</td>
<td>LoadSetLaunchProcessFromAttachment.java</td>
</tr>
<tr>
<td>Task Properties:</td>
<td>processPropName: A property containing a process which is loaded, set, then launched.</td>
</tr>
</tbody>
</table>

## Load process from URL

<table>
<thead>
<tr>
<th>Description:</th>
<th>Loads a process from a URL [deprecated]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability in IDE:</td>
<td>Task list navigator, under: processes # start tasks</td>
</tr>
<tr>
<td>Implementation Package:</td>
<td>org.dwfa.bpa.tasks.process</td>
</tr>
<tr>
<td>Implementation Class:</td>
<td>LoadProcessFromURL.java</td>
</tr>
<tr>
<td>Task Properties:</td>
<td><strong>ProcessDataId:</strong> A data id for the process container to load. Only data containers that contain I_EncodeBusinessProcess objects can be dropped. <strong>processURLString:</strong> A URL from which a process is loaded.</td>
</tr>
</tbody>
</table>
The Business Process Library comprises a set of business processes that support the terminology development environment of an organisation. Business Processes are constructed using the Business Process IDE as described in *Automating Business Processes*, and saved to the analyst's local disk. Business Processes that are to be run by modellers or other end users should normally be incorporated into the IHTSDO Workbench IDE distributed to users' machines. Other Business Processes, e.g. those supporting administrative activities such as configuring modeller's environment, may not be built into the distribution bundle.

A Business Process (BP) utilises the plugin framework of IHTSDO Workbench and could be configured to be invoked through several means, by choosing the appropriate location in the IDE bundle folder/directory structure:

- A BP can be invoked using a button in IHTSDO IDE. The button can be attached to one or more of a number of panels in IHTSDO IDE.
- A BP can be invoked from the IHTSDO IDE menu.
- A BP can be loaded into the business process editor from the user's filesystem and executed therein. This is useful for administrative processes undertaken by analysts, such as configuring a particular user's modelling or validating environment.
- A BP could be invoked by the system as a subprocess of a larger Business Process.

For example, the precommit and commit folders store business processes that are executed as part of the commit action within the IHTSDO IDE.

The mechanism for BP invocation is controlled in the IHTSDO IDE by the corresponding serialised BP object (a file with extension .bp) built by the Business Process Editor being located in an appropriate subfolder of the "plugin" folder in the executable bundle. This chapter tries to describe all of the above Business Process types.
The Business Process Catalogue

The following pages describe each of the business processes that have already been constructed. A BP is generally considered a plug-in if it resides in one of the specialised locations in an IHTSDO IDE bundle. These locations then determine how the BP can be invoked. Rather than categorising the BPs by their plug-in type, we have chosen below to order the processes by their business function category, dealing with change sets, creation processes, refset processes, etc. However, the plug-in type is also included in each table. There are also a number of business process modules that are loaded and invoked manually by an analyst, usually to establish, configure or modify a modeller’s editing environment. These BPs are not usually considered plug-ins, nor are they usually supplied with the application bundle.

Change Set processes

<table>
<thead>
<tr>
<th>Manually import all changesets for user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
</tr>
<tr>
<td>Rationale:</td>
</tr>
<tr>
<td>Availability in Editor:</td>
</tr>
<tr>
<td>Plug-in type</td>
</tr>
<tr>
<td>Implementation Package:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manually import all changesets for user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
</tr>
<tr>
<td>Rationale:</td>
</tr>
<tr>
<td>Availability in Editor:</td>
</tr>
<tr>
<td>Plug-in type</td>
</tr>
<tr>
<td>Implementation Package:</td>
</tr>
</tbody>
</table>

Creation processes

<table>
<thead>
<tr>
<th>Add user account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
</tr>
<tr>
<td>Rationale:</td>
</tr>
<tr>
<td>Availability in Editor:</td>
</tr>
<tr>
<td>Plug-in type</td>
</tr>
<tr>
<td>Implementation Package:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Create profile folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
</tr>
<tr>
<td>Rationale:</td>
</tr>
</tbody>
</table>
All the settings are saved in the profile, including the editing path, root concepts and refset setting. The folders for saving the business processes which can be process or queue, and changesets.

### Availability in Editor:
- **Plug-in type**: manual
- **Implementation Package**: Create Profile Folder.bp

### Create new concept from viewer

**Description:**
This process will replace the default 'new concept' process in the IHTSDO Workbench bundle. The default 'new concept' process create a new concept as the child of selected concept in taxonomy view. When creating new concept in ED project, user's requirement is that create a new concept from concept viewing tab. So this process is created.

Firstly the process will find the "active" concept tab which is user viewing, then create a new concept in this tab and add a relationship new concept-is a -viewing concept in this tab.

### Availability in Editor:
- **Plug-in type**
- **Implementation Package**: Create new concept from viewer.bp

### Pop up SCT relationship type

**Description:**
Clear the profile setting "rel types for pop up"

Find the top category of the editing concept, find all available relationships type in the model. Then put those relationships into "rel types for pop up"

**Rationale:**
When users are creating a new SCT concept in IHTSDO IDE, there may be a requirement to adhere to the concept model relevant to that part of the SNOMED hierarchy. For example, when editing concept is a "clinical finding" the model will limit the relationship types during adding relationships.

This business process populates the appropriate relationships according to the concept model.

### Availability in Editor:
- **Plug-in type**
- **Implementation Package**: Pop up SCT relationship type.bp

---

**Processes for Statistics**
## Set up root for user and save

**Description:**
This process will set the root for terminology editing to the appropriate top level concepts in the IHTSDO Workbench hierarchy. These may include:

- local Auxiliary concept tree,
- Refset auxiliary concept tree,
- Terminology auxiliary concept tree,
- SNOMED CT concept tree.

This process clears the taxonomy root setting in the user’s profile, and adds the appropriate tree roots as top level concepts. It then saves the user’s profile.

**Rationale:**
Modellers and Quality Assurance validators need access to concept hierarchies (e.g. the relevant SNOMED CT release), and other editing concepts. Each of these represents a distinct hierarchy, and each and every hierarchy needs to be navigatable within the editor in a given session. This process ensures that the required hierarchies are loaded and available in the editor’s hierarchy navigation window and that concepts in each are available for use in other editing functions, via UUIDs.

**Availability in Editor:**
manual load only

**Plug-in type**
manual

**Implementation Package:**

---

## Bundled Business Process Plugins

The following BPs are undergoing testing and enhancement at the time of writing.

### Load and Classify

**Description:**
Loads and runs the default classifier on a RefSet.

**Rationale:**
It is important that the SNOMED refset is self-consistent with the SNOMED description logic and that it adheres to the specific concept model constraints and rules. Furthermore, the inferred view of the refset, denoting the derived relationships inherited from higher concepts of a given concept needs to be available during the modelling phase. Satisfying all the above requirements is enabled by running an appropriate classifier over the refset.

This process loads and runs the classifier and can be invoked manually (via button) by a modeller, as well as being available for autoprocessing.

**Availability in Editor:**
Can be invoked through the SnoRocket Button.

**Implementation Package:**
LoadAndClassify.bp

---

6§
## Subversion Processes

### Synchronize with Subversion

**Description:**
This process will synchronize the user's work with the project development subversion server.

The IHTSDO IDE will update the user's folder with the server UPL which is stored in the user profile, then commit any change with the comment.

**Rationale:**

**Availability in Editor:**

**Plug-in type**

**Implementation Package:** Synchronize with Subversion.bp

## Conflict Resolution

### Display Conflicts

**Description:**
This process will display conflicts associated with a given concept arising from either dual independent modelling, or failures to validate against editorial/design constraints.

**Rationale:**
The existing IHTSDO IDE conflict display is too generic to meet the needs of the terminology project and needs to be extended to support the additional constraints and editorial rules pertaining thereto.

**Availability in Editor:**
toggle button in concept panel bar

**Plug-in type**

**Implementation Package:** yet to be implemented

### Inferred vs Stated View

**Description:**
This process will toggle the concept display view between the SNOMED stated view, and the inferred view as determined by the classifier.

**Rationale:**
The existing IHTSDO IDE toggle button is merely a placeholder.

**Availability in Editor:**
toggle button in concept panel bar

**Plug-in type**

**Implementation Package:** yet to be implemented
Processes define business processes as a series of tasks, determine the execution sequence of tasks, and record the execution history of tasks. Processes are also tasks, and can therefore encapsulate processes as tasks that can be contained within other processes.

In the IHTSDO Workbench IDE, the Process Builder provides a way to create or build reusable processes or tasks that can be executed within different processes. For example, a process to add a new user or incorporate a concept can be used in different processes, even though the processes are designed to accomplish different results.

The IHTSDO Workbench Launcher Queue holds reusable processes for execution. The original process remains in the queue after a duplicate of the process is launched. This allows users to launch a particular type of process, such as adding a new user, many times. Other processes may control more discrete functions, such as determining the permissive level of a check. User interactions can be managed by the process layer of IHTSDO Workbench as componentized, reusable assets that present information to - and collect information from - end users.
To review the Process Builder, from the IHTSDO Workbench IDE window, click on the Process Builder icon, as shown in . The window shown in will display where the Concept Work Area is usually visible.

The Process Builder Window contains four primary sections:

- Task Directory
- Task List
- Process Header
- Process Work Area

Task Directory

The Task Directory is similar to a folder or directory on your hard drive. It displays all of the tasks available to you when creating processes. Items in the task directory can contain one level or multiple levels. Click on the lever on the left of a task directory to display sub-levels associated with that directory. For example, clicking on the workbench task directory will display the sub-directories shown in .
The task directory is a permanent component of the IHTSDO Workbench IDE. On a PC, use Windows Explorer to view the folder where tasks and processes are stored. For example, the directory/folders shown in illustrate the folder hierarchy where IHTSDO IDE saves processes and tasks on the hard drive.

**Task List**

Each task directory contains a list of tasks specific to that directory. To view a task list, click on any directory. For example, when you click on the address-sub-directory in the IHTSDO IDE tasks directory, the window will display the list of tasks shown in.
Process Header

The process header contains descriptions that uniquely identify the process. These fields are explained in more detail in *Completing the Process Header*.

- **Priority**: Indicates, from lowest to highest, how the process should be prioritized in the assigned queue.
- **Process ID**: A unique identifier generated by the application.
- **Process Name**: A descriptive name assigned by the process originator.
- **Subject**: A variable text field that further defines the process, such as a concept name.
- **Originator**: Usually the email address of the process originator. This field is not used if the process will be executed locally (on a single computer).
- **Destination**: Routing if the process is to be electronically delivered to another workflow environment. This field is not used if the process will be executed locally. This field is currently disabled as a security feature.
- **Attachments**: Used to indicate there is an external component, such as a form item, related to the process.
- **Add Attachment Key**: Specifies the attachment of key (form item key in a form).
- **View**: A drop-down menu that changes the process diagram view between history, tasks and messages.
- **Current Task ID**: The active task within the process.

Process Work Area

The process work area is the space where you can build and change processes by selecting tasks from the task directory and creating the required workflow by linking the selected tasks.

Using the Process Builder

The Process Builder provides a powerful graphical mechanism to construct workflow process templates that can support and enact an organisation's business processes. A sample business process is illustrated in . Once constructed, a business process can be saved, loaded, reedited, and executed as and when required. Each business process is built up from a set of interconnected tasks. The tasks must already exist in the IHTSDO Workbench IDE bundle. Each task performs a unit of work. Each task in the standard task library is described in detail in Chapter *Automating Business Processes*.
A business process can invoke another business process. In this sense, the second business process is often referred to as a *subprocess*. Subprocesses just appear as tasks in the Process Builder. They are differentiated by having their name underlined, and the word *process* appearing next to the task Id in the task block in the process layout pane.

Each task is implemented by a piece of java code, either written generically for the Editor, or written by a software developer/analyst to meet specific local requirements. Each task is usually implemented by a single java class file (together with a corresponding BeanInfo class file). Each business process is merely a collection of tasks supplemented with some metadata and flow data. There is no corresponding executable code required to be written to support the business process independent of its constituent tasks. Thus the core of much of the functionality required is contained in the special java files that constitute the task library(ies), and hence the importance of *Automating Business Processes*.
The set of (some 200+) tasks available for constructing a business process or workflow is organised by folder and displayed in the task navigator on the left of the Process Builder. The top level categories of tasks are:

- IHTSDO Workbench- basic editor functions, including gui and concept related tasks.
- log - tasks to log actions undertaken in an edit session
- sim - simulation tasks
- web - tasks to interact with web sites
- grid - tasks to support synchronous multiuser activities
- queue - tasks to manage instances of business processes and execution queues
- deadline - timer tasks to ensure processes complete
- flow tasks - tasks associated with process flow, including branching, completing and aborting process instances.
- processes - task associated with managing process instances, including launching, getting/setting properties, etc.
- processes - collection of prebuilt business processes that can be used as provided, or as subprocess tasks in large process.

Note that there appear to be two folders labelled "processes". The first of these contains tasks for controlling a process, such as loading or launching a process, or setting/copying a process property. The second “processes” folder contains potential subprocesses - tasks which are themselves whole processes. Any process that is saved via the save process button is available, by default, under this folder. These special process tasks appear in the Process Builder with their names underlined as shown in .
Building a new business process involves selecting one of the tasks for the new process from the task navigator and dragging and dropping it onto the Process Diagram canvas. This is repeated for each of the component tasks. Each task will be given a task Id, unique within the business process. If the same task is required twice within the same business process, then each use will be given a unique Id. The task Ids are assigned during the process flow definition phase, which involves chaining the tasks together to form a complete workflow. This is done task pair by task pair, using the mouse by linking a pair’s downstream task’s title to the continue ID slot in the upstream task. This then populates the upstream Continue Id with the Id of the downstream task, as shown.

When creating a new process, the Process Builder automatically places the two tasks Change Instance Id and Set Deadline onto the process diagram pane. Additional tasks are selected from the task catalogue and dragged onto the diagram pane. They can be positioned by mouse (+Shift key) or arrow keys.

The completed Business Process can then be tested with the Execute button, and saved on the filesystem as a .bp file (serialised javabean), for later reuse.

**Exercise - Creating a Business Process**
This exercise will help to illustrate process builder concepts by guiding you through the steps of creating a simple process, starting from the Process Builder window.

From the Process Builder window, click on the Process Builder icon immediately above the task directory. The process diagram panel will be pre-loaded with two tasks: “Change Instance ID” and “Set Deadline”.

The “Set Deadline” task represents a deadline and priority in a queue by using time increments. Processes are executed in priority and deadline order, similar in concept to the date received and priority stamps on an email message. The deadline and priority are meaningless until the process is sent to a queue.

Click on the misc tasks directory in the task directory list.

In the task list, click and drag the “Show Dialog” task into the Process Diagram panel, as shown in .

**Repositioning a Task Box in the Process Diagram**

To reposition a task box in the process diagram, hold down the Shift key while you drag the box to its new position.

The Show Dialog task box includes a message area. Type “Hello World” in the message area.

**Linking Tasks in the Process Diagram**

Note that each task box displays an ID number, as in “id: 0” in the Change ID task box. This number controls the order in which tasks in the process will be executed. Processes must always start with ‘id: 0’ in the first task box.

To link the Show Dialog task to the process, click on the “Show Dialog” heading in the task box, then drag and drop it on the zero (0) in the Continue box in the Set Deadline task box. The process diagram should look like .
Note that the “Id” numbers of the tasks are now in consecutive order from zero to two and the blue arrow is connecting the “Set Deadline” task to the “Show Dialog” task.

**Testing the Process**

To test the “Hello World” process you just built, click the Execute button in the lower right corner of the window. Your screen should look like the one shown in.

**Adding a Task to the Process**

When you execute the “Hello World” process, and then click OK when the “Hello World” message displays, you may notice a message in red type at the bottom of the window after the “Process Complete” message in blue type. This message indicates that the “Show Dialog” task did not have a branch, or another task, following it.

To complete the process, click on the flow directory and click and drag the “Process Complete” task to the process diagram panel, as shown in.
Connect the “Process Complete” task to the “Show Dialog” task by clicking and dragging the red Process Complete heading in the task box to the zero (0) in the “Continue” box in the “Show Dialog” task. The process diagram should now look like the one shown in.

Click the Execute button to run the process and click OK when the “Hello World” message displays. Note that the message at the bottom of the window displays only “Process Complete” without any conditional information.

## Saving a Business Process

Before saving a process, you will need to complete the process header. The process header includes the following fields:

**Priority:** Click on the drop-down menu to set the priority level you want the process to assume within its assigned queue.
Process id: This is a unique identifying code generated by the IHTSDO Workbench application.

Process Name: Delete the generated name and enter a name that makes the process quickly identifiable. For the example, type Hello World and press Enter.

Subject: Enter a description that more fully identifies the process, such as a patient’s name.

Originator: An optional field if the process will be executed from a local (your) computer. Enter your email address or another type of address.

Destination: An optional field if the process will be executed from a local (your) computer. If the process will be sent electronically to another workflow environment, you must identify the routing and delivery process here. This field is currently disabled as a security feature.

Attachments: Check this box if you are adding either an attachment or a key.

Add Attachment: Click Add Attachment to attach an external document, such as a patient record. The Add Attachment dialog will display, as shown in .

![Add Attachment](image)

Locate and click on the file you want to attach and click Open. The file you selected will be displayed in a table immediately below the Add Attachment button. For example, attaching a patient record will look similar to the example shown in .

![Attached Files Table](image)

Add Key: Click Add Key to attach an item with a “null” or unknown value. The key will retrieve the value from the designated location. For example, you may need to attach a form item, such as a blood pressure reading, to a process so the value (the blood pressure reading) can be checked against acceptable ranges and, if out of range, trigger an alert on the form. Please refer to Exercise: Creating a Process With a Key for an example of adding a key.

**Saving the Process**

When you have completed the process header, from the File menu, click Save Process. The Save Process dialog shown in will display. Navigate to the forms folder in the IHTSDO Workbench folder (processes/workspace/IHTSDO Workbench/form), or other folder where you store processes.
Enter the process name without any blank spaces - for example, helloworld or hello_world. Click Save. The IHTSDO Workbench IDE will add the .bp file extension.

**Loading a Business Process**

To load an existing process, from the Process Builder File drop-down menu, click on Read Process. The Open Process dialog shown in will display.

Navigate to the “Hello World” business process you saved and click Open. The process is loaded into the Process Diagram panel ready for execution.

**Executing a Process**

Please refer to *Adding a Task to the Process*.

**Exercise: Creating a Process With a Key**
Now that you have learned how to create a simple process such as “Hello World,” this exercise will walk you through the steps required to build a more complex process and one that includes adding a key.

These processes will provide a way to display an alert on a form for blood pressure values outside of acceptable ranges.

From the Process Builder window, click on the File drop-down menu and click New Process. The window will look as shown in . Your screen may look slightly different, depending on the last task directory you viewed.

**Creating a Number Check Alert**

To check blood pressure readings against a pre-defined range, use the Number Check Alert task.

Click on the IHTSDO Workbench IDE task directory and the alerts sub-directory. The alerts task list will display, as shown in .
Click and drag the Number Check Alert task to the process diagram panel, as shown in . Position (using Shift/drag) the task box where the entire task is visible.

Link the Number Check Alert task to the process by dragging and dropping the blue Number Check Alert title on the task box to the zero (0) in the Set Deadline task box Continue field, as shown in .

Note that the blue arrow now connects the “Set Deadline” task to the “Number Check Alert” task and the Continue field on the “Set Deadline” task has changed from zero to 2.

**Completing the Number Check Alert Task**

The Number Check Alert heading is displayed in blue. This indicates the task is a process embedded in a task. The task box contains additional steps. Note: An underlined task name indicates the task has an embedded process.

Alert Text: The Alert Text box provides an area to specify the alert message you want displayed on the form. Delete the default text and type “Systolic blood pressure greater than 160 Hg/mml”.

Warning: You can specify one of three alert levels: informational, warning or error. Click on the drop-down menu arrow and click on “Warning”.

Number CT Value: Enter a number that indicates the value that should trigger the alert. For this exercise, enter 160.

Comparison: The Comparison field provides a way to specify the parameter of the value entered in the Number CT Value field: Equal, greater than (GT), greater than or equal to (GTE), less than (LT), less than or equal to (LTE). Click on the drop-down menu arrow and click on GT (greater than).
Numeric Item Key: This field is already filled with the default value A: FORM_ITEM, which indicates the numeric value to be checked will reside in a form item.

When you have finished entering and selecting options for the Number Check Alert task, the task box will look like the one shown in.

![Number Check Alert Task Box]

### Completing the Process Diagram

To fully complete the process diagram, add the “Process Complete” task from the flow task directory by dragging and dropping the task box from the task list to the process diagram panel, as shown in.

![Process Complete Task]

Link the Process Complete task to the Number Check Alert task by clicking and dragging the red “Process Complete” heading to the False: 0 field on the Number Check Alert task. Then click and drag the red “Process Complete” heading to the True: 0 field on the Number Check Alert task. The diagram now should look like the one shown in.
Linking both the False and True Task IDs to the “Process Complete” task indicates that the process should end, regardless of whether the systolic blood pressure value is greater than 160 Hg/mml.

Note that the “id” numbers in the task boxes are in consecutive order, from zero to 3.

**Completing the Process Header**

Before you can save the process, you need to identify it for easy retrieval. Complete the fields as follows:

- **Priority:** Normal
- **Process Name:** “Systolic Blood Pressure Check”
- **Subject:** “Checking for high blood pressure”
- **Originator:** An optional field. If you wish you can enter your email address.
- **Destination:** Not required since the process will be executed on your (local) machine.
- **Attachments:** Click in the box to add a checkmark. A table will display below the field.

**Adding an Attachment Key to a Process**

Adding a key to a process provides a way to attach a Form Item (as found in a patient form) that has a null or unknown value. The process will retrieve the value as calculated in the form, for example, a patient’s systolic blood pressure reading. With this methodology, the IHTSDO Workbench application can perform a number check against the value specified in the process (the Number Check Alert task). If the number retrieved from the form item is outside the specified range, an alert will be displayed in the form while it is in use.

To specify where the Systolic blood pressure check process should retrieve the blood pressure value, click on the Add Key button. A record will be added to the table, as shown in .

To correctly identify the form item key, you must use the same label (name) that is used in the process diagram. In this example, delete the type “FORM_ITEM” in the name field of the table. Press Enter to confirm the change.
Exporting the Attachment Key

The process must export the attachment key you have identified so the request can be passed to the form when a provider uses it. To export the key, click on the tasks drop-down menu in the lower portion of the tasks list. Click on Properties, as shown in.

The task list at the top of the window will change to a Task/Property list, which includes the FORM_ITEM. Click in the box next to FORM_ITEM, as shown in.

Saving the Process

From the File drop-down menu, click Save Process.

Navigate to the folder where you save processes and enter a file name, being careful not to leave any spaces in the name. For example, type systolic_blood_pressure_check and click Save.

Test Yourself: Create Another Process

Following the instructions in the section Exercise: Creating a Process With a Key, create another process that checks for diastolic blood pressure greater than 95 Hg/mml.

Using the Terminology Viewer in Processes

Processes can include terminology from the Terminology Viewer to, for example, place a process in a queue. To use a terminology concept, add a process step that incorporates a concept that can be derived from the Terminology Viewer.

To begin, add a task to the process from, for example, the queue tasks, by double-clicking on the queue task in the task list. The queue tasks directory will display as shown in. Drag and drop (or cut and paste using Ctrl/C and Ctrl/V) a queue task into the process and link it to the task above (if appropriate).
In the example above, the queue task “To Queue” was added to the process. The process will need a specific queue type destination in order to send the process to the appropriate queue. You can retrieve the queue type from the Terminology Viewer.

For ease of use, divide the display screen so that you can view the Terminology Viewer and the Process Builder windows at the same time.

In the Terminology Viewer, locate the terminology concept, such as Queue Type, and click on the lever to open the list of queue types. Drag and drop the desired queue type, such as “inbox queue” to the box under “Queue Type” in the process diagram, as shown in .

Viewing Process Diagram History

You can test processes by pressing the Execute button in the lower right corner of the window. To view a history of the process’s execution, click on the View drop-down menu above the process diagram panel and click History. The screen will look similar to the example shown in .
The history view shows when each task in the process was executed and the result. Tasks are initially shown in task id order. Click on the Date column to view tasks in date order.
Integrating the SnoRocket Classifier

The SnoRocket Classifier is a highly optimised module for performing complete or incremental classification of SNOMED CT.

SnoRocket Structure
Creating Snorocket in Eclipse
Integrating with the Workbench
Build
Create a Workbench Process to Classify
Incremental Classification
Code Notes
Gap Analysis
SnoRocket Structure

The source build is structured using a Maven parent project structure. The snorocket-parent project aggregates 5 modules.

**snorocket-parent**: The Maven parent module. Nothing but the POM.

**snorocket-core**: au.csiro.snorocket.core The core algorithm implementation, at the level of the underlying EL+ description logic rather than being SNOMED-specific.

**snorocket-snapi**: au.csiro.snorocket.snapi An abstraction layer over the Core to provide a set of SNOMED-centric APIs and post-processing (SNAPI == SNomed API)

**snorocket**: au.csiro.snorocket A command-line harness including parsers for various file formats (SNOMED distribution tables, KRSS, etc)

**snorocket-bridge**: au.csiro.snorocket.ace The bridge code that connects Workbench's I_SnorocketFactory to a concrete implementation of au.csiro.snorocket.snapi.I_Snorocket

**snorocket-tests**: au.csiro.snorocket A batch of test input files and some code to check that snorocket produces output isomorphic to the SNOMED distributions (the isomorphism part is because group numbers are not preserved, nor are relationship IDs or file ordering).

Creating SnoRocket in Eclipse

Prerequisites: Subclipse, m2eclipse

Repository URL: [https://csfe.aceworkspace.net/svn/repos/ihtsdo-classifier](https://csfe.aceworkspace.net/svn/repos/ihtsdo-classifier)

From the SVN Repository perspective, add the repository and do “Checkout as Maven Project...”

Create a Maven build configuration, with goals “clean install” for the snorocket-parent project.

Run it.

Integrating with the Workbench Build

Modify the Workbench POM by adding `snorocket-bridge` as project dependency and also a dependency for the dwfa-mojo plug-in, as follows:

```xml
<dependency>
  <groupId>au.csiro</groupId>
  <artifactId>snorocket-bridge</artifactId>
  <version>1.1.2</version>
  <scope>compile</scope>
</dependency>
```

Modify `src/main/assembly/lib-bundle.xml` and add the following includes to `/lib`:

```xml
<include>au.csiro:snorocket-bridge</include>
<include>au.csiro:snorocket-core</include>
<include>au.csiro:snorocket-snapi</include>
```

Note that the Snorocket pom file needs to reflect the correct ACE API version.
Create a Workbench Process to Classify

Use the tasks NewClassifier (org.dwfa.ace.task.classify.NewClassifier) followed by Load&Classify (org.dwfa.ace.task.classify.LoadClassifyWrite) as shown in

Note that writing the inferred relationships back into the Workbench database is not part of the existing Workbench Snorocket integration.

Incremental Classification

To get the incremental classification of Snorocket into the Workbench bundle modify the pom file to include the following, which makes the incremental classification capability defined in the ClassifyCurrentConcept business process available as a plug-in.

<extractAndProcessSpec>
<filePatternStr>^org/dwfa/ace/plugins/component/ClassifyCurrentConcept.bp</filePatternStr>
<destDir>ace-bundle.dir/plugins/component/</destDir>
</extractAndProcessSpec>

The incremental classifier business process classifies the concept currently being edited against the base state of SNOMED and displays the results in the signpost. In the business process shown below, it can be seen ( ) that the base state for classification is maintained as a file on the local file system. This file is in the root directory of the Workbench install.
For SNOMED, the size of the file is 195MB. This is a plain text file, with the classification state written as a printed representation. It is likely that the file size could be reduced considerably if it were serialized as a binary object.

Note that the business process creates the classification base state file if it is not present. For SNOMED, in this situation, memory is exhausted when attempting to load the base state in a 32-bit Java environment. This appears to be due to the initial classification data structures not being released for garbage collection after the file is written. It takes slightly more than 5 minutes from the time the incremental classification is initiated until the out of memory error is encountered. Restarting enables the capability to function.

The diagram shows the results of an incremental classification.
Note that incremental classification does not permit editing base SNOMED concepts.

The function of the incremental classification has some limitations. For example, creating a primitive sub-concept and classifying yields no results in the display. There is also no display of equivalent concepts. No provision is made for detecting or displaying concept cycles (C1 isa C2 and C2 isa C1). Similarly, role cycles are undetected.

**Testing**

Snorocket includes a number of small test cases which are exercise specific aspects of the classifier. These are organized in the JUnit framework. Doing the Maven build and including execution of tests exercises the test suite. In addition to a number of traditional unit tests, this also includes exercising the classification of the January 2009 SNOMED stated relationships, which could be considered a system test from the Snorocket SNOMED perspective. The time required to do the Maven build including the test suites is slightly more than 10 minutes.

**Performance**

The Snorocket classifier was tested on the January 2009 release distribution of SNOMED using the newly available stated relationships release. The timings include reading the SNOMED input files from the file system and writing the results. The system configuration was Mac OS 10.5 on 2.8 GHz Intel 2 core with 4GB memory.

Total time was 163 secs.

Broken down into the three main phases, including elapsed time and memory footprint (following GC):

- Load: 8 secs, 145 MB
- Classify: 83 secs, 653 MB
- Write: 72 secs, 721 MB

These performance results may be obtained on other platforms by running the test with the system property "log.memory" set to "T". ("-Dlog.memory=T"). Inspect the log for entries of the form "Used memory @ ... " and "Time @ ...". The relevant entries are "Post loadOntology", "Post classify", and "Post xxxRelationships".
Role Hierarchies

SNOMED make use of role hierarchies such as "Indirect device" as a sub-role of "Procedure device". In the DL formalism these are expressed as role inclusions. If "Indirect device" -> "Procedure device" then (some "Direct device" "Arterial stent") -> (some "Procedure device" "Arterial stent") See section 4 in the SNOMED User Guide. Also, navigate (in the Workbench) as follows:

Linkage concept / Attribute / Concept model attribute

The ISA hierarchy under "Concept model attribute" reflects the role hierarchies Snorocket uses these ISA relationships to define the role inclusions. Specifically if a concept exists in SNOMED which also appears as a role relationship, then the parent of that concept is the right hand side of the role inclusion axiom for that concept. Thus, "Indirect device" ISA "Procedure device"

results in

"Indirect device" -> "Procedure device"

A consequence of this is that "Concept model attribute" appears in concept inclusion axioms even though it never appears as a role. See snapi.Snorocket.Populator.processRoleRow

Right Identities

There is currently no provision in the Snorocket Workbench interface for defining right identities. The current January 2009 SNOMED release has one defined per the Perl script which generates KRSS format from the stated forms table:

direct-substance o has-active-ingredient -> direct-substance

The possibilities for incorporation into Snorocket include:

Call addRelationship(direct-substance, has-active-ingredient, direct-substance) and then modify the else block in processRoleRow of snapi.Snorocket.Populator.

Extend the snap.I_Snorocket interface with addRightIdentity(r1, r2, r3), and update Poplater to do the appropriate ontology.add(new RI(new int[] {r1, r2}, r3)) call. This either requires hardcoding in the Workbench Snorocket interface or determining a representation in the Workbench.

One possibility for representation of the right identity within the current release format would be to create a "Right identity" concept in the attribute hierarchy similar to the "Isa" concept. This could then be used to create a relationship among the attribute concepts. So the concept "Direct substance" would have a "Right identity" relationship to "Has active ingredient". An example of this is shown in
**Gap Analysis**

**Writing Inferred Relationships into the Workbench database**

The inferred relationships derived by the Snorocket classifier are not written back into the Workbench database. This is an important aspect for modellers to be able to view. This could be accomplished by writing these relationships with a characteristic type of "inferred". A somewhat more compact form would note the characteristic type as "inferred and defined". The latter may require additional editing logic in the UI, for example to remove a defining relationship but still note its inferred status. Perhaps the Workbench path mechanism would mitigate this.

Once the inferred relationships are in the Workbench database, it would also be useful to provide notification to the user that classification is necessary. That is, visibly noting that the inferred relationships do not necessarily reflect the current definitions. This sort of capability might operate similarly to the commit button, by enabling a "classify" button whenever definitions change since the last classify. Note that this is a property of the state of the db, not the UI.

**Displaying Equivalent Concepts**

The presence of equivalent concepts in SNOMED is not necessarily an error from the perspective of the description logic formalism, but does present an error in modelling from the SNOMED perspective. If two concepts were equivalent in SNOMED their descriptions should be merged for example.

There should therefore be a prominent display to the user that equivalent concepts exist and a means to display the sets of equivalent concepts for remediation.

**Displaying Cyclic Definitions**

Modellers may also introduce cyclic definitions, for example, stating A isa B and B isa A. Note that these may also be introduced via role relationships and may be of arbitrary length. These are indeed errors and should be detected and displayed for remediation.

**Right Identities**

Some means needs to be devised to handle the definition of right identities.
Remediation of Incremental Classification Defects

As noted earlier there are several aspects of the incremental classification capability that need to be addressed. These include exhausting memory when creating the classification state, display of concept equivalence, and detection of cycles. There are also ways in which the results display could be improved. Some progress information should also be displayed during the initial creation of the classification state.

Improved Memory Utilization

When classifying SNOMED, the Snorocket memory footprint is fairly substantial. Since Snorocket and ACE were independently developed, there is some duplication of information. On 32-bit environments this could become a substantial issue as SNOMED grows. One way to minimize memory footprint is by running Snorocket in a separate process. If the incremental classification capability has utility for extensions, certain environments may benefit by maintaining a classification engine for base SNOMED and classifying just the extension in a servlet container or continuous integration environment. A related area is the file size for the classification state, which could likely be reduced by writing as a binary object (as opposed to the current plain text).
Using the Queue Viewer

The Queue Viewer in the IHTSDO IDE provides a centralized location for launching processes, as well as to store reusable processes, to schedule processes to execute at a specified time, and to archive processes not in current use or that have already been executed. Processes can be queued by department or any defined work group or activity.

Accessing the Queue Viewer
Touring the Queue Viewer
Sorting the Process List
Viewing a Process in a Queue
Understanding the Process Diagram
Moving a Process to Different Queues
Accessing the Queue Viewer

From the IHTSDO Workbench IDE window, click on the Queue Viewer icon, which is shown in .

The Queue Viewer will display in the area of the window where the Component Work Area typically is displayed, as shown in

Touring the Queue Viewer

The Queue Viewer is designed to replicate the look and functionality of an email browser. The viewer contains three primary sections, as shown in .
Queue Name List

The Queue Name list on the left is similar in function to a list of unopened email messages.

Understanding Queue Types

Processes are executed by workers, who generally retrieve them from queues. Queue types include:

- **Aging Queue**: Holds processes that must be aged for a specified time period before continuing execution, such as a process that must wait for results of lab work.
- **Archival Queue**: Stores unused or completed processes.
- **Inbox Queues**: Several of the queues displayed in are inbox queues used in the simulation example. They include C&P Eye, GME, Eye Care and PCP. In some cases, processes will be held in the inbox queue until a user selects them for action. In other cases, the inbox may be associated with workers that automatically select and execute processes based on criterion such as priority and deadline.
- **Compute Queue**: Holds processes awaiting execution based on their priority and deadline. Processes placed on the compute queue must not require human interaction for task completion.
- **Launcher Queue**: A queue from which processes can be started or launched.
- **Outbox Queue**: Holds processes for delivery to another location by unspecified means, usually via media or some network.
- **Sync Queue**: Holds processes that must wait for another process to complete before it runs. For example, if a patient must see the doctor, the process cannot be executed before the nurse and diabetic educator see the patient. The processes must sync up to ensure the proper sequence of events. Methods such as SMTP or FTP.

Queue Reading Area

The area on the right, the Queue Reading Area, (which is initially empty when IHTSDO Workbench IDE is loaded) provides a space for you to “read” the processes stored in a queue.

To better understand this functionality, click on the Launcher Queue to open the Queue list, as shown in .

Process Diagram Viewing Area

The Process Diagram Viewing Area provides a way to display the process diagram associated with a process selected from the Queue Reading Area. See .
**Sorting the Process List**

The process queue list will initially display in Process ID order. The queue list can be sorted by any of the column headings to group processes according to similar characteristics. For example, click on the Subject column to sort processes by subject. Any processes without a subject will appear at the top of the list, followed by all other processes in alphabetical order according to the subject text, as shown in .

<table>
<thead>
<tr>
<th>Name</th>
<th>Subject</th>
<th>Deadline</th>
<th>Priority</th>
<th>Originator</th>
<th>Process ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Initial Workspace</td>
<td>Demonstrate grid process</td>
<td>292276994.08 18 2</td>
<td>Normal</td>
<td>[email]</td>
<td>644597f8e-deb2-4ea...</td>
</tr>
<tr>
<td>Set Ferro, Master Cl</td>
<td>Current EHR</td>
<td>292276994.08 16 2</td>
<td>Normal</td>
<td>[email]</td>
<td>6c48010e-8e1b-412...</td>
</tr>
<tr>
<td>Set Ferro, Tony, Jr</td>
<td>Current EHR</td>
<td>292276994.08 16 2</td>
<td>Normal</td>
<td>[email]</td>
<td>5f7a277-c827-4a8...</td>
</tr>
<tr>
<td>Break Password</td>
<td>Demonstrate grid process</td>
<td>20083.11 11 20 32 37</td>
<td>Normal</td>
<td>[email]</td>
<td>e625f4e-4abc-548...</td>
</tr>
<tr>
<td>Increment Master Cl</td>
<td>Increment Master Cl</td>
<td>292276994.08 16 2</td>
<td>Normal</td>
<td>[email]</td>
<td>203509492-4dc...</td>
</tr>
<tr>
<td>Increment Worker Cl</td>
<td>Increment Worker Cl</td>
<td>292276994.08 16 2</td>
<td>Normal</td>
<td>[email]</td>
<td>2535f6e-5e29-441...</td>
</tr>
<tr>
<td>Forest Master Cl</td>
<td>Forest Master Cl</td>
<td>292276994.08 16 2</td>
<td>Normal</td>
<td>[email]</td>
<td>e625f4e-4abc-548...</td>
</tr>
<tr>
<td>Reset Worker Simul.</td>
<td>Reset Worker Simul.</td>
<td>292276994.08 16 2</td>
<td>Normal</td>
<td>[email]</td>
<td>89f9d6ec-008c-45b...</td>
</tr>
<tr>
<td>Put Chart On Server</td>
<td>Selected from file cl.</td>
<td>292276994.08 16 2</td>
<td>Normal</td>
<td>[email]</td>
<td>654676b8-910c-491...</td>
</tr>
<tr>
<td>Set Worker Simul.</td>
<td>Set Worker Simul.</td>
<td>292276994.08 16 2</td>
<td>Normal</td>
<td>[email]</td>
<td>559f6a-008c-43f...</td>
</tr>
<tr>
<td>Set Worker Sched.</td>
<td>Set Worker Sched.</td>
<td>292276994.08 16 2</td>
<td>Normal</td>
<td>[email]</td>
<td>218567f-5e29-447...</td>
</tr>
<tr>
<td>Test move with fail</td>
<td>Test process errord</td>
<td>292276994.08 16 2</td>
<td>Normal</td>
<td>[email]</td>
<td>2c13e5e-3c29-305...</td>
</tr>
</tbody>
</table>

**Viewing a Process in a Queue**

To view a process in a queue, click on a process row in the list. For example, click on the “Break Password” process and the process will display below the queue reading area. To “hide” the queue list so you can view the entire process diagram, click on the up arrow above the diagram area. The screen will look like the one shown in .

You can also use the entire window to display and view a process by clicking on the left-facing arrow next to the Queue Name heading as shown in .
The full screen view of the process is shown in.

Understanding the Process Diagram
While all of the fields and components of a process are also described in the chapter Exercise - Creating a Business Process, it is important to understand the functions that are also available within the Queue Viewer.

Priority: Indicates, from lowest to highest, the process’s priority in the assigned queue.
Process ID: A unique identifier generated by the IHTSDO Workbench IDE.
Process Name: A description name assigned by the process author.
Subject: A variable text field that further defines the process, such as a patient’s name.
Originator: Usually the email address of the process originator. This field is not used if the process will be executed locally (on a single computer).
Destination: Routing if the process is to be electronically delivered to another workflow environment. This field is not used if the process will be executed locally.
Attachments: Used to indicate that there is an external component, such as a form or a patient record, related to the process.
Add Attachment: Specifies the attachment, such as a patient record.
Add Key: Specifies a key, such as a FORM_ITEM, used to pass a value, such as blood pressure reading, to the process so it can be checked or validated.
View: A drop-down menu that changes the process diagram view between history, task, and messages.
Current Task ID: The active task within the process.
Process Diagram: Please refer to the chapter “creating processes with the Process Builder for a detailed explanation of process diagrams.

Moving a Process to Different Queues

Processes can move to different queues without manual intervention or user selection. For example, a “Jumping Bean” process can move from its existing queue to a different queue specified by the process as shown in .

To see how the process moves from queue to queue, build the process diagrammed in , based on the information you previously covered in the chapter Automating Business Processes. (Hint: You will need to find the “To Selected Queue” task in the queue task directory and copy and paste the “inbox queue” queue type from the Terminology Viewer.) Then execute the process and note that it moves from the existing queue to the inbox queue.
This chapter is aimed at those software developers who intend to programatically extend an IHTSDO Workbench IDE (e.g. editor or viewer). Those considering to do so should be both proficient in the Java programming language, and be well versed with the existing concepts and functionality described in this manual. Experience with automatic software build processes in a collaborative environment would also stand the prospective developer in good stead. This, and the following chapters which constitute the Developers Guide, are only intended to be sufficient in detail to guide prospective developers, rather than be a comprehensive developers’ resource. Those wishing to embark on extending the functionality of IHTSDO Workbench are advised to peruse the available source code and discuss their proposed changes or enhancements with those in the international community who have trodden a similar path before them.
Extending the IHTSDO Workbench IDE programatically

The IHTSDO Toolkit is based on a predominantly open source infrastructure and built for maximum configuration and customisation by users, to suit their needs and to enable collaborative development of terminologies. Other features of this tooling environment are that it can be adapted, extended, enhanced, and that features and functionality available to users can be re-used, and re-purposed, without having to rely upon vendor capacity, service level agreements, or business development priorities and interests.

Although software developers may have their own preferred development methodologies and environments, the approach covered here has evolved and been refined to optimise productivity and distributed development across different operating systems. The key technologies employed are Java; JINI; Eclipse; Maven; Subversion; a versioned object database built on Berkeley DB; install4j; and the IHTSDO Workspace and Maven internet repositories.

Key Concepts

Bundle: a collection of artefacts associated or “bundled” together for a single purpose – distribution and installation on customers' desktops. Bundle can also be used as a verb.

Package: a collection of associated artefacts “packaged” into a single jar file for version management purposes.

module: [Maven concept]

project: [Maven concept] is the smallest unit of software that is built and packaged as a single artefact, with its own build instructions, using Maven. The build rules for the project are represented in the project's POM, and contain all of the project's goals and dependencies.


dependency: [Maven concept]

goal: [Maven concept]

repository: [concept used by both Maven and Subversion]. A persistent store for holding various sorts of artefacts in a controlled fashion. It supports multiple users, version management, web access. A subversion repository on Collabnet may be used for storing source code for developers of IHTSDO Workbench IDE components and Maven plugins. A Subversion repository may also be also used by terminology modellers for storing terminology changesets and change history from their IHTSDO Workbench profiles. Maven repositories are used to hold built components (jar files) that are used for constructing executable IHTSDO Workbench IDE bundles, replete with SNOMED and/or other terminology sets within a single database.

deploy: [Maven concept] The action of transferring a project's built artefacts from a developer's local Maven repository into a remote master repository.

install: [Maven concept] A project goal which, when executed, places a built project artefact into the user's local maven repository. Maven checks all project dependencies and, if required, will automatically download or build relevant components of the project.

Snapshot: [Maven concept] Unlike stable releases, which are deemed immutable and stay in one or more Maven repositories until no longer supported, snapshots can change daily, or even hourly to reflect developer's changes.

Continuous Integration: When a team of developers is working on a project, the contributions of each member can conflict with those of the others on the team. By frequently integrating and testing individuals' changes into a single system, the risk of divergent or conflicting change is minimised. Organisations can use the IHTSDO

---

8 Java Archive – a single archive file format from Sun Microsystem's Java initiative. Similar to a zip or other archiving technology. Jar files contain a MANIFEST section of metadata about the archive.

9 http://www.collab.net
Workbench Continuum\(^\text{10}\) server to drive Maven builds after source code changes have been committed to the IHTSDO workspace.

**Development Principles**

This section articulates the key principles underpinning the terminology development environment. The development environment comprises the hardware and software infrastructure as well as aspects of the rendering of a terminology development organisation's business processes using this infrastructure.

- Continuous Integration
- Automatic Builds
- software and hardware

**The Terminology Modeller's Environment**

Terminology modellers need access to various versions of a given terminology. They need tools that support the importation of termsets, the automatic correlation of a given term in there termset of interest to SNOMED or other target terminology. They need the ability to edit the descriptions associated with a given concept, the relationships to other concepts, the ability to deprecate terms, and a host of other functions that the IHTSDO Workbench editor is capable, or might be capable of providing. A simplified view of the modeller's environment is illustrated in Figure 172, “IHTSDO Workbench Terminology Modeller’s Environment”, below. It is important to note that the environment is distributed in both time and space, with modeller's having the ability to work offline and to synchronise there changes with those of others through a master repository that holds changesets. Each modeller's changeset is timestamped and identified as belonging to that modeller. This allows for changes to be uploaded in chunks appropriate to the work practices of them modeller, and for changes to be rolled back if required. It also supports the important principle of dual independent modelling which helps to improve the quality of a terminology.

---

\(^{10}\) http://continuum.apache.org
**The Terminology Validator Role**

The role of the validator is to analyse the potential updates and changes devised by the terminology modeller(s) and resolve any conflicts that have arisen as part of the Quality Assurance framework. Terminology development is undertaken using a dual independent modelling approach, whereby each target concept is modelled in SNOMED by two modellers working independently. The change sets produced by the two modellers are then compared and presented to the validator role. At the time of writing, tool support for this process is still being developed.

**Setting up Eclipse for IHTSDO Workbench Development**

The Eclipse IDE has been used extensively for programatically extending the IHTSDO Workbench components. Eclipse is a cross-platform development environment, particularly suited to java based applications and available from the Eclipse Foundation\(^\text{11}\). Two particular Eclipse plugins that improve productivity for IHTSDO Workbench development are the Maven 2 plugin and the Subversion plugin available from Eclipse Plugin Central\(^\text{12}\).

Once these plugins are installed, they will need to be configured to support the connection to the IHTSDO workspace. For subversion, this may require editing the servers file in the developer’s subversion directory to support proxy connections, if required. For Maven, the settings.xml file in the developer’s local maven repository will need configuring to support authentication to the IHTSDO Workspace repository.

An overview of components involved in the distributed development of the IHTSDO toolkit is illustrated in

\(^{11}\) [http://www.eclipse.org](http://www.eclipse.org)  
\(^{12}\) [http://www.eclipseplugincentral.com](http://www.eclipseplugincentral.com)
The next chapter *About the IHTSDO Workspace*, explains the pivotal role of the IHTSDO Workspace in the distributed development environment for extending IHTSDO Workbench functionality.
About the IHTSDO Workspace

The IHTSDO Workspace is comprised of three CollabNet products: CollabNet SourceForge Enterprise Edition (CSFE), CollabNet CUBiT and CollabNet Subversion. These three products all work together to provide the robust, secure collaborative platform that is required to support terminology development organisations’ members and licensees in the collaborative development, advancement, and delivery of terminologies such as SNOMED CT worldwide.

With more than 1.2 million users, CollabNet is the most widely used platform for Distributed Software Development. CollabNet transforms the way software is being developed by simplifying distributed development and enabling organizations to leverage global development talents to deliver better products and innovate faster.

Using the CollabNet platform, worldwide project members that are geographically distributed can work as “one team” throughout the lifecycle of a project, regardless of their location. Unlike traditional development tools, CollabNet supports globally distributed teams in a cost-effective way, providing savings through reduced development infrastructure costs, increased productivity, improved project visibility and shared access to software assets.

More than 500 companies rely on CollabNet for their distributed development, off-shoring, outsourcing and partner co-development efforts.

Founded upon open source principles, CollabNet is also the company behind Subversion, the next-generation Software Configuration Management solution. Subversion was named sole leader in standalone SCM in the Forrester Wave report for Software Configuration and Change Management, Q2, 2007.

The CollabNet platform reduces development, build and test infrastructure costs by enabling central support of geographically distributed teams and eliminating the need to replicate or install local software asset repositories. Many CollabNet customers have cut their development infrastructure costs by up to 50 percent.

CollabNet products enable collaboration, which improves transparency and productivity through project dashboards and integrated wiki, forum, mailing list, and document management tools that support global search, role-based security, and universal web/e-mail/RSS access.

With CollabNet, development teams become more efficient and collaborate more openly. Many CollabNet customers have experienced reduced time-to-market, delivered error-free projects ahead of schedule, and increased their market leadership.

In summary, the IHTSDO Workbench workspace will allows organisations and their members and licensees to:

Rapidly gain visibility, traceability and governance across terminology development and the tools and processes used to create a terminology release
Unify the terminology development lifecycle tools and processes into a common, integrated platform
Accelerate terminology development and innovation
Accelerate creation of and distribution of terminology development tools and processes
Provide a common terminology development platform
Eliminate the time spent configuring servers and reduce costs through optimizing servers across a global distributed pool
Reduce terminology build cycles and increase quality by providing a continuous integration and release environment
Manage costs at each stage with comprehensive rollup reports by server, user, and project
Provide certifiable quality and traceability for governance and change management through comprehensive audit logs, dashboards, and reporting
Publish a release

IHTSDO Workbench provides end-to-end support for distributed development and application lifecycle management, and allows integration of this end-to-end support directly into Integrated Development Environments such as Eclipse.
Integrated and On-Demand

There are other options for each of the components provided by the IHTSDO Workspace, and some of those options are available on demand as a service. One area that sets the IHTSDO Workspace apart from other offerings is the level of integration between all the components, and being able to access this integration on-demand by a group wholly dedicated to service and support of distributed and collaborative development environments.

If the on-demand applications are not integrated, users will become weary of jumping from one application to the other to access the data they need. For example, a worker who uses on-demand issues management software, collaboration software, and version control software, would find they have to move in and out of different environments to perform different tasks.

Accessing these applications from various providers on the Internet will become taxing on workers who crave an easy to use common platform, accessible by anyone, at anytime, to conduct all of their business-critical operations.

By using an integrated CollabNet platform, workers can reach beyond the confines of their offices with anyone they need to do business with - customers, partners, and suppliers - all without any barriers.

In addition, the CollabNet platform can authenticate against an external single-sign-on provider, allowing future integrations that span provider boundaries as required.

Configuration Management—CollabNet Subversion

IHTSDO Workbench depends on CollabNet Subversion for version control and configuration management. CollabNet makes Subversion available as a service, enabling globally distributed organizations to share version-controlled resources across locations. Ease of use and out-of-the-box support for remote teams make Subversion the best solution for global projects, compared to legacy tools that are inadequate for distributed teams and too expensive to run.

Subversion was named sole leader in stand-alone SCM by Forrester Research for the reasons above, as well as its best of class version control capabilities, low-cost support for distributed teams, and enterprise-class scalability. By delivering certified binaries, commonly needed add-ons, support, and services, CollabNet Subversion minimizes the risks associated with deploying an open source solution in the enterprise.

Users from remote development sites can access IHTSDO Workbench’s web-accessible central Subversion server, and eliminate the need for repository replication, additional local servers, and system administrators. Developer productivity is also increased by enabling web-based access to a wide range of modern SCM techniques, including trunk-based branching and merging, continuous integration, and nonexclusive file locking.
Subversion continues to be the fastest growing open source application for version control - installed on nearly 300,000 public Apache servers as of July 2008 and used by well over one million developers.

CollabNet Subversion is an enterprise-ready distribution of Subversion that includes certified binaries, platform specific installers, certified plug-ins for other tools, and enterprise-ready add-ons.

An overview of CollabNet Subversion 1.5 benefits include:

- State of the art Software Configuration Management
- Applying versioning to files, as well as directories and file meta-data.
- Support branching and tagging operations efficiently and independently of the branch or tag size.
- Guaranteeing integrity through true atomic commits—nothing takes effect until the entire commit has succeeded
- Handling binary files efficiently.
- Supporting trunk-based branching, continuous integration, and nonexclusive file locking.
- Tightly integrating Subversion with IDEs.
- Centralized repository
- Providing access to repository from any local or global network.
- Facilitating artifact management, traceability of changes, and governance.
- Sharing artifacts between projects, enabling project reuse.

The following sections describe the benefits of Subversion 1.5 in more detail.

**Sparse Checkouts**

Users can choose to check out only part of the source tree from the repository to use as their local working copy. This means that they no longer have to check out all of the files – they can check out only what they need.

**Repository Mirroring**

Teams that are geographically dispersed often experience slow response time when hitting the central server. In order to improve response time, Subversion allows a local copy, or mirror, of the repository that is used for all “read” operations. The repository mirror is kept in sync with the central repository. All write operations will still go through to the central server using a WebDav write-through proxy.
This feature provides the benefit of a centralized repository for version control with distributed teams, but the benefit of local response times for read operations.

**Repository Sharing**

For people using the flat file repository for Subversion, the system will automatically limit the number of files kept within a single folder. For example, Subversion can now allow 1000 files to be saved in a particular location before creating a new location automatically. This “sharding” makes it easier for third-party tools, such as an Explorer type program, to find and manage files.

**Directory Versioning**

Directories, renames, and file metadata (but not timestamps) are versioned. Entire directory trees can be moved around and/or copied very quickly, and retain full revision history.

**Atomic Commits**

This means that Commits are true atomic operations. Interrupted commit operations do not cause repository inconsistency or corruption.

**Global Revision Numbers**

Whereas most version control systems assign a separate number to each change to each file, each Subversion revision number defines a complete, coherent set of versions of all files and directories in the entire repository. This means that every revision number constitutes a complete "change set" or "baseline", guaranteeing simple assurance of "which changes go with which."

**Cheap “branches” and “tags”**

Subversion branches, tags, and copies are "cheap" both in the cost to create them, and the cost to store them in the repository. Making a tag, branch, or copy costs the same time and space regardless of whether it affects one file or ten million—and that cost is very low, around a second of time and a few bytes of space.

This low cost is a dramatic contrast to most other version control systems, which can take hours to make these structures. Since all version control systems need some sort of lock while doing this, these other systems can cause user operations to fail simply because some other user was making a tag; not so in Subversion.

**Versioned Properties**

Subversion itself attaches "properties" to files and directories, such as the fact that a file is executable. Subversion also defines other properties that you can set on your files and directories, which cause Subversion to treat them specially. For example, the "MIME type" lets Subversion clients know whether to display a file using a word processor, a graphic editor, or some other tool.

Subversion allows you to define your own properties, which can mean anything you want, typically used in tools you build on top of Subversion. For example, you might define a property that connects a particular change to a file with the new test case which was added to verify the change. All of these properties are "versioned" along with the files: if the value of the property needs to be changed over time (such as the connection to the test case mentioned above), this happens automatically.

**Locking**

There are really two kinds of locking: "internal locking" used to protect Subversion itself (which was mentioned in the above discussion of tags), and "user locking," which is implemented and enforced by Subversion, but provided to the users for their own purposes. The most common use of locking is to prevent anyone else from making changes to a certain file, perhaps because you're planning some big change and don't want to deal with conflicts.

**Efficient Delta Handling**

"Deltas" are the result of comparing a version of a file to the previous version—just the changes this time around. They're used pervasively throughout Subversion, saving disk storage, saving network bandwidth, and helping to
manage multiple changes to the same file (if one delta is on line 10, and another is on line 100, we know they don't interfere with each other).

Subversion's deltas are unusually "efficient" in several ways:

Since they're used everywhere, they actually have to be computed less often than in many systems

They're used with binary files as well as text files (unlike many version control systems), which is all the more important since binary files tend to be much larger than text files

Perhaps most importantly, Subversion uses deltas in both directions, whereas most version control systems can use them from the server to the user, but not in the other direction

**WAN-Readiness/Offline Operations**

There are several aspects to Subversion's WAN-readiness; offline operations are but one:

Subversion's data transfers between client and server (that is, over the WAN) are deltified whenever possible

Subversion's operations are streamed: requests for more data are being sent while the earlier requests are still being satisfied, so there are no pauses between chunks

Subversion ensures that many key common developer operations can be performed "off-line," without any need to contact the server at all:

Find your changes to your current set of files

Compare your changed version to the version you got from the server

Revert your changes back to the server's version

---

**Layered API Library**

Subversion's code was carefully designed in "layers," so that other programs can use whichever parts they need. As a result, alternate client implementations (like Subclipse, Tortoise Subversion, and SCPlugin) can use exactly the same code as the basic "svn" command line tool, and be confident they will receive the same results and features.

**Remote Repository Sync'ing**

Subversion's "svnsync" tool can copy changes out of a master repository and store them in a slave or secondary repository, so the two always have the same data.

Subversion 1.5’s new "WebDAV write-through proxy" achieves a similar effect in the other direction, ensuring that changes made in the secondary repository are also copied up to the master. Together, they enable fully synchronized repository replication. This can provide enhanced performance for geographically distributed teams, as well as disaster recovery features.

---

**Application Lifecycle Management — CollabNet SourceForge**

IHTSDO Workbench uses CollabNet SourceForge Enterprise® (CSFE) for application management. Built around Subversion, SourceForge is the leading platform for globally distributed software development. From anywhere in the world, all team members can securely access and manage the source code, issues, releases, documents, discussion forums, wikis, reports, and other artifacts related to their projects.

Using SourceForge, administrators can centrally provision all users and assign project asset and tool access permissions based on project role. They can also automate and manage processes with custom workflows, audit logs, and automated monitoring and notification features.

CollabNet SourceForge Enterprise is an integrated suite of Web-based SCM, issue tracking, project management, and collaboration tools that empowers teams to build escalation and notification features so stakeholders can be proactively notified of user-specified events such as overdue tasks and status changes. By centralizing management of users, projects, processes, and assets, CSFE dramatically reduces costs, increases productivity, and improves project visibility.
CSFE was created for improving and enabling distributed development and is built around Subversion. CSFE improves productivity with the integrated software configuration management, file release management, issue tracking, and project management tools such as task and tracker reports.

CSFE enables sharing of information through discussion forums, mailing lists, and documentation management that support versioning, role-based access and universal search.

CSFE was developed to improve productivity and drive reuse with an integrated suite of development and collaboration tools, securely manage projects with centralized access controls and custom workflows, improve project governance with end-to-end lifecycle management, and leverage existing tools, expertise, and assets with the industry’s most open and flexible platform.

CollabNet SourceForge Enterprise 5.0 offers numerous benefits to developers and project managers.

By coordinating the entire lifecycle with complete Application Lifecycle Management (ALM), CSFE improves visibility and establishes end-to-end traceability. It accomplishes this by associating source code with requirements, issues, tasks, documents, discussions and other artifacts.

Further, CSFE models the ALM process with customizable project pages and dynamic portlet-like components. Its project templates allow users to capture and share workflows, best practices, and other project content to promote process standardization.

CSFE simplifies management and security with common user and project administration. It supports easy modeling and enforcement of organizational structures and relationships (such as outsourced development or QA teams, partner co-development) with a rich role-based access framework. With CSFE, you can map specific tool- or resource-access levels (for example, bug and feature databases, source code repositories, mailing lists, wikis, document folders, or file releases) to the needs of a particular role or group to comply with security policies. In addition, you can validate users against existing LDAP systems for centralized access control.

CSFE provides tools to easily create and manage project workspaces, allowing you to standardize or customize project resources, content, and organization. The entire site can be customized to align community and corporate identities.
CSFE enables distributed SCM with Subversion and file release systems by establishing a secure, central artifact and project content inventory. As a result, it simplifies SCM repository creation, configuration and browsing. Issues, requirements, and other artifacts are associated with commits in one or multiple SCM repositories. These capabilities allow you to publish and manage approved file release packages easily and confidently.

Issues and other development artifacts are tracked and managed with CSFE’s integrated change management. With this functionality, you can establish workflows between any artifacts, including requirements, feature requests, defects, issues and tasks. Stakeholders receive automatic notification of changes.

With CSFE, project management visibility is improved, allowing greater insight into project plans, deliverables and development status. Activity metrics support real-time evaluation of project activity. Data is summarized into convenient dashboard views of graphical reports and trends.

Collaborating with integrated forums, mailing lists, wikis, and document management is streamlined. CSFE supports global searching across all project communications. You can capture and archive forum and mailing list history and automatically generate a knowledge base. CSFE’s ability to maintain a complete history and versioning of documents and wikis provides team members with valuable information. In addition, document reviews – including comments, approvals, and publications – are easily initiated and managed. CSFE enables instant notification of changes to monitored discussions, wikis, documents, or artifacts.

Lastly, CSFE project tools enable interoperability and extensibility with Open Web Services APIs.

CSFE Tool Integration – CSFE provides a single, common view into the data held by all tools that are integrated with the CSFE platform. This includes CSFE tools, and third-party developed tools. Integration is achieved through pre-built CSFE tool integrations or via well-documented APIs and the CSFE SDK that supports SOAP XML Web Services and Java RMI APIs.

Enterprise-class architecture

CollabNet SourceForge Enterprise Edition is built using a highly scalable and extensible Java 2 Enterprise Edition (J2EE) architecture. The scalability of CSFE has been proven with several real-world deployments of over 10,000 users. It features a comprehensive, open API based on SOAP web services together with a complementary SDK to facilitate easy integration and extensibility with external systems and tools. The ability of SourceForge to integrate seamlessly with external technologies in a straightforward and non-disruptive fashion is instrumental in protecting and extending investments in existing tools, training and expertise.

Three-Tiered Web Services Oriented Architecture

CSFE was architected and built explicitly for enterprise integration and extensibility. This will allow terminology developers to use this framework to integrate with many different partner solutions. The CollabNet SDK is downloadable from the www.open.collab.net website, available for all customer and partners.
Role Based Security

CSFE helps protect all assets via secure, role-based access controls and operations. These access controls allow authorized personnel to set secure, fine-grained permission on what users can see, access and change at all levels, both project and tools.

SourceForge Explorer

CSFE also includes a .NET application that communicates via SOAP API with CSFE.

Project Management

CSFE integrates bi-directionally with Microsoft Project (and other project management tools). Project plans created in MS Project can be easily loaded into CSFE, allowing staff and managers to assign tasks, track status, rapidly identify problems and dependencies, and maintain a complete project audit trail within CSFE, together with the ability to re-sync data with MS Project at anytime.

Project Dashboard

Staff at every level of the organization (subject to their role and permissions status within CSFE Enterprise Edition) can open their browser and view the high-level, real-time status of every project. They can instantly see which projects are on track and which are not and where problems exist. Project managers will spend less time gathering data and looking for problems and more time solving them. CSFE even provides automated monitoring, escalation, and notification features so stakeholders can be proactively notified of user-specified events such as overdue tasks, status changes, and more.

Collaboration

Collaboration tools and capabilities within CSFE will make it easier for team members and organizations to work together, no matter where they are located. Favorited collaboration methods such as email are supported in CSFE via mailing lists, list management and the archiving of all e-mail in threaded discussion forums. In addition, CSFE provides discussion forums, centralized news, and project status reporting that helps keep everyone on the same page. Support for asynchronous, written collaboration methods such as email and discussion forums are particularly useful for distributed organizations. Time-zone differences sometimes make phone calls and instant messaging impractical. Emails and forums give people time to research and understand issues, frame responses, and ask clarifying questions.

Project communications and other project artifacts (documents, issues, tasks, code, etc.) may be associated with each other inside CSFE. This allows relevant links to be made between materials submitted by different users and also helps team members quickly understand the context of issues and activities to which they are assigned. They spend less time “reverse-engineering” information from multiple sources and systems and more time making productive contributions.

Document Management

CSFE includes an easy-to-use document management system that enables users to store, share, collaborate, and manage documents in virtually any file format. The system includes a straightforward, configurable workflow mechanism for managing document review, approval, comment, and publication. It also provides full document versioning and comment history. All documents stored in the CSFE Document Manager are indexed for the purposes of searching and can be associated with one or more project artifacts to add context and clarity.

Requirements Management

CSFE has native tool support for Requirements Management. This enables project/applications requirements to be developed, refined, documented, and managed within a common development platform. Requirements that are in the “formative stages” in MS Word documents and spreadsheets can be managed with the CSFE document manager until their subsequent entry into the requirements management system.

Developer Management

As a common global development platform, CSFE provides the underlying workspace, or “backbone”, that unifies application project activity regardless of the type of IDE used by individual developers. Developer access to versioned assets, discussion forums, mailing lists, and centralized build and test environments can all be managed through the IHTSDO Workspace.
Change Management

As a global development platform, CSFE integrates with a wide range of change management systems. We are recommending CollabNet SourceForge as the repository foundation. However, CSFE can integrate with other change management systems.

“ALM” Templates

CSFE supports the use of project templates as blueprints to quickly create new projects with pre-defined project structure, content, Tracker artifact types with embedded workflows, and Tracker queries.

This feature includes being able to import and customize ALM templates when creating new projects to establish local or global process and workflow standards.

Issue Tracker

SourceForge includes a sophisticated, highly configurable, easy-to-use issue tracker. The Tracker is general purpose and can be configured for automatic capture, management, communication and resolution of defects/bugs, requirements, change requests, incidents, support requests and more.

Help

Online help is available for all CSFE tools. CollabNet also offers many web based training modules that will be available.

Build, Test and Publish Services Management—CollabNet CUBiT

IHTSDO Workbench uses CollabNet CUBiT\(^\text{14}\) to enable distributed development teams to quickly access a pool of on-demand build and test services. Teams can manage their own library of continuous build software stack profiles, rapidly provision that configuration onto an available machine, and version control the profile throughout development, build, and QA testing.

CUBiT significantly reduces cost and time: Configuring servers for the code, build, and test cycle is 10 times faster, and teams save up to 90% of hard costs by centralizing development tool, server, and administrative infrastructure.

Using CUBiT, teams can quickly configure test and build tools, stacks, and servers when they need them. They eliminate the time spent on configuring servers and save costs through optimizing servers across a global, distributed pool. Reuse of servers and profiles improves software quality while reducing administrative and hardware costs. Teams can use a standardized stack for their project and share across projects to eliminate up to 50 percent of the time spent on configuring servers.

CUBiT provides a searchable build library for build results, artifacts, and components can be shared with others to support continuous integration. By providing access to and management of the global pool of servers, server use can be optimized and costs are controlled. CUBiT enables administrators to manage costs at each stage with comprehensive rollup reports by server, user, and project. Audit logs, dashboards, and reporting ensure traceability for governance and change management.

CollabNet CUBiT enables development teams to access a global pool of on-demand code, build, and test services. CUBiT eases the server-intensive requirements of methodologies such as agile, scrum, branch based development, unit testing, and continuous integration. Development teams are able to accelerate development cycles, eliminate build and test errors, and gain flexibility in utilizing their machines - to more easily adapt their development environment to iterative and collaborative approaches.

The IHTSDO workspace utilizes CUBiT to build the tools, documents, and the terminology release, and to publish the results of the build. This is accomplished through the use of several open source software products: Maven, Continuum, and Archiva.

CollabNet CUBiT 2.0 benefits include self-service, dynamic allocation of development, build and test services that accelerate development with on-demand build and test tools, application stacks, and servers available. Developers can fulfill their own needs by drawing from a pool of standardized development, build, and test resources. You can further extend the pool of resources by building your own cloud with CollabNet, your own, or Amazon EC2 cloud resources.

\(^{14}\) CUBiT is an acronym for Centralized Unified Build, Integration and Test.
Centralized management of builds and test software stacks translates to less time spent setting up and configuring systems. Version controlled software stack profiles are managed as a secure library for consistency across the development, build and test cycles. Project managers can create custom continuous integration profiles or choose from standards based on CruiseControl or Maven. Test tools profiles can also be created to meet custom requirements.

CollabNet CUTiT 2.0 provides centralized management and reuse of build components, thereby eliminating productivity loss due to broken builds and errors from mis-configuration. Build processes are standardized with a library of build and test stacks. Continuous integration is made possible by automated end-to-end process and workflows.

Project control across a global pool of computing resources, projects, profiles and build artifacts is facilitated with visibility into the continuous integration status. Access to software artifacts and hardware assets is secure and audit trails record each request. Projects costs of assets can be tracked from various clouds, and cost accounting of machine time for project and user reporting is also logged. Role-based permissions via CSFE control the level of each user’s access.

Provisioning

CollabNet CUBiT supports provisioning of Physical Hosts for Intel/AMC, Sun Sparc, VMware Server, VMware ESX, Solaris Containers, and hosts via API. Also supports Eclipse IDE integration, user virtualization management, cloud services for Amazon EC2 and CollabNet, private cloud management and cloud services via an API plus self-servicing, with on demand provisioning.

Profile Library

Profiles can be selected, created and edited, permissions added and metadata attached. Also available is profile administration, version control, cost accounting for each profile. There is a centralized profile library management with automatic profile creation from an image snap shot.

Cost Accounting

Usage is tracked and accounted for based on software stack cost and CPU per hour cost.

Reporting

Reporting includes download build logs and files, upload build logs and files, downloadable user activity, project activity and system activity. Reports for full system monitoring by node and performance of the nodes are also available.

RBAC

CUBiT supports the same Role-Based Access Control (RBAC) permissions from CSFE so project membership can be shared across both products.

Deployment

CUBiT can be hosted by CollabNet or onsite at any customer location. A shared managed instance is also available.

Integrations

CUBiT integrations include Maven, Cruise Control, Continuum, Ant, BuildForge, Electric Cloud, the Eclipse plug-in and the Microsoft Visual Studio plug-in.
**Smart-Client Integration**

IHTSDO Workbench uses the CollabNet Integration Framework (CIF) to make configuration management, application lifecycle management, and build and test services management available to smart clients. CIF provides programmatic access to functionality and data within the CollabNet Platform. The CIF allows CollabNet customers and partners to build integrations and extensions that support application lifecycle management using the CollabNet Platform.

The CIF includes a collection of SDKs featuring Web Services APIs, integration components, developer guides, code samples and support forums. CIF resources are available for these CollabNet products:

- **CollabNet Subversion** is extensible through APIs and language bindings that enable server and client side extensions and integrations.
- **CollabNet SourceForge Enterprise** is extensible through a comprehensive set of SOAP Web Services APIs that provide access to the SCM, Change Management, Project Management, User and Project Administration, Collaboration, and ALM capabilities of the platform. CollabNet SourceForge Enterprise user interface can be changed through velocity templates.
CollabNet CUBiT is extensible via REST-based APIs that allow developers to integrate their applications with CUBiT’s capabilities to manage infrastructure for software development.

Using CollabNet SourceForge and CUBiT, smart clients are integrated into a collaborative workspace that supports the entire terminology lifecycle. Cubit is used to host:

Maven and Continuum – Maven and Continuum provide the basis for “continuous integration”, or essentially a continuous build process applying quality checks, generating workflow items and producing reports with the goal of iteratively improving the content’s quality. Maven controls the build process and provides a framework for executing quality tests and reports, while Continuum schedules constant execution of these processes triggered by modification to software or data contained in the distributed application.

Archiva – Archiva is an application for managing one or more remote repositories, including administration, artifact handling, browsing and searching.

Each smart-client has access to these components and the services that they provide. The combination of these smart clients communicating over a Subversion based bus, together with the CUBiT-hosted Maven, Continuum, and Archiva provides the full functionality of the distributed application.
Creating new tasks

Example Task Extensions
Steps to Create A Task

The IHTSDO Workbench Editor's functionality can be extended through building business processes based on predefined tasks. Although there are over two hundred separate tasks already available as components for a business process, there may well be a requirement for a developer to create or modify one or more tasks for building any given business process. This chapter gives guidance on how to accomplish this.
Example Task Extensions

There are several simple tasks that can be used as the basis of extending the workflow environment by adding your own tasks. Here are some:

1. au.com.nch.bpa.tasks.web.FetchFromWeb A task that reads a string from a URL, parses the string, and presents the string to the user in a dialog. This task demonstrates how tasks can connecting to servers over the network, and do something with those results.

2. org.dwfa.bpa.tasks.dialog.ShowInfoDialog A task that displays a message to the user. The message was set when the process was created in the process builder, using standard JavaBeans property editor conventions.

Steps to Create A Task

There are three steps to creating a task:

1. Create the task class by extending org.dwfa.bpa.tasks.AbstractTask. Put the code to execute the tasks desired action in the evaluate method of your class. Pattern your class after one of the existing example tasks, then add new functionality as desired. The org.dwfa.bpa.tasks.web.FetchFromWeb task and the org.dwfa.bpa.tasks.dialog.ShowInfoDialog tasks provide simple examples.

2. Create a BeanInfo class for your task. This BeanInfo class controls how the task is presented in the process builder, and how its properties are edited within the process builder. More information about how to program the BeanInfo class can be found at here. Your BeanInfo class must be named following standard conventions where the first part of the name is identical to the name of your task class, followed by "BeanInfo". Your BeanInfo class must be in the same package as your task class.

3. Add an entry to the pom.xml to write the task to disk as a JavaBean, and include this task in the ed-sct-bundle application bundle so it can be accessed from the process builder. The generation of JavaBeans during the Maven build process is managed by the dwfa-maven-plugin module. The syntax for including a new JavaBean specification for this plugin is to add a beanSpec entry such as this entry for the FetchFromWeb task:

```xml
<plugin>
  <groupId>org.dwfa.maven</groupId>
  <artifactId>dwfa-maven-plugin</artifactId>
  <version>1.0-SNAPSHOT</version> [change as necessary]
  <configuration>
    <specs>
      <beanSpec>
        <sourceName>org.dwfa.bpa.tasks.web.FetchFromWeb</sourceName>
        <dirName>web</dirName>
      </beanSpec>
      <specs>
    </configuration>
    <specs>
  </plugin>
```

Additional bean specifications can be specified by adding additional beanSpec entries to the plugin configuration.

Step by Step Example

The following steps will create a simple "hello world!" task from within eclipse:

[15] see http://java.sun.com/j2se/1.5.0/docs/api/java/beans/BeanInfo.html
1. If you have not already done so, follow the steps described in Eclipse Integration to import the process module into your eclipse environment.

2. Within eclipse, select the org.dwfa.bpa.tasks.dialog package. Right click on this package, and select New > Class. Name the new class HelloWorldDialog, and set the Superclass to be org.dwfa.bpa.tasks.AbstractTask, and select the checkbox labeled "Inherited abstract methods". Now commit this dialog, and create the class.

3. Change the method:

```java
public int[] getDataContainerIds() {
    // TODO Auto-generated method stub
    return null;
}
```

This method will be deprecated in future versions of the architecture.

4. Change the method:

```java
public Collection<Condition> getConditions() {
    // TODO Auto-generated method stub
    return null;
}
```

This method defines the possible valid exit conditions for this task. In this case, we just want the environment to continue with the next task after executing this task, hence the CONTINUE_CONDITION specification. Other tasks may allow for branching conditions, or other exit conditions, and may define a different collection of exit conditions.

5. Add the following fields and methods to control the serialization of this task:

```java
private static final long serialVersionUID = 1;
private static final int dataVersion = 1;
private void writeObject(ObjectOutputStream out) throws IOException {
    out.writeInt(dataVersion);
}
private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException {
    int objDataVersion = in.readInt();
    if (objDataVersion == 1) {
        //all is well :-)  
    } else {
        throw new IOException("Can't handle dataversion: "+ objDataVersion);
    }
}
```

Although this task is very simple, other tasks may have more complicated data structures which are set at design time using standard JavaBean editing conventions. By implementing the serialization methods, and by explicitly
defining the serialVersionUID and dataVersion, we are providing a foundation for supporting task evolution over time.

6. Change this method:

```java
def public Condition evaluate(I_EncodeBusinessProcess process, I_Work worker) throws TaskFailedException {
    // TODO Auto-generated method stub
    return null;
}
```

to the following:

```java
def public Condition evaluate(I_EncodeBusinessProcess process, I_Work worker) throws TaskFailedException {
    JOptionPane.showMessageDialog(new JFrame(), "Hello World!");
    return Condition.CONTINUE;
}
```

The task is now fully functional. To use the task from within the Process Builder, (see the Clinic Demonstration for an introduction to running the bundled environment, and the Process Builder), we must create a BeanInfo class for this task.

7. Create the BeanInfo class by extending java.beans.SimpleBeanInfo. Within eclipse, select the org.dwfa.bpa.tasks.dialog package. Right click on this package, and select New > Class. Name the new class HelloWorldDialogBeanInfo, and set the Superclass to be java.beans.SimpleBeanInfo. Now commit this dialog, and create the class.

8. Add the following method to this class:

```java
def public BeanDescriptor getBeanDescriptor() {
    BeanDescriptor bd =
        newBeanDescriptor( HelloWorldDialog.class);
    bd.setDisplayName("<html><font color='green'>
                  <center> Hello World</center>
                </html>");
    return bd;
}
```

Now that the Task has a BeanInfo class to describe it, it can be made available to the process builder by writing the task bean to disk. This step is managed by adding an entry to the Maven pom.xml file as follows.

9. Open the process pom.xml file from within eclipse, and add the following entry below the entry for org.dwfa.bpa.tasks.dialog.ShowInfoDialog (you can use eclipse to search the file to find this entry):

```xml
<beanSpec>
  <sourceName>
    org.dwfa.bpa.tasks.dialog.HelloWorldDialog
  </sourceName>
  <dirName>dialog</dirName>
</beanSpec>
```

10. Generate a new "bundle" using maven by navigating to the dev/bundle directory and executing either mkbundle.sh (unix) or mkbundle.bat (windows). This command causes all the modules to be built, and bundled in a directory. This directory will be located inside the dev/bundle/target/dwfa-bundle.dir/dwfa when the build is complete. Go stretch for awhile :-) It takes 40 minutes to do this complete bundle build on our development machines.
11. Once the bundle build is complete, you can execute the resulting bundle by executing the startClinic.sh (unix) or startClinic.bat (windows) file located inside of the dev/bundle/target/dwfa-bundle.dir/dwfa directory.

12. After launching the clinic bundle, you should be able to locate the new task from within the Process Builder by selecting the "dialog" folder in the upper left panel of the window, and the new task should be displayed in the lower left panel, and can be dragged onto the process panel on the right.
An IHTSDO Workbench IDE application bundle is a package of components ready for installation on an end-user's computer. Such a bundle includes the various software components, configuration files, policy files, predefined workflow tasks, a database containing one or more versions of SNOMED, and other additional term sets relevant to the terminology modeller's functional requirements. A terminology modeller may have several different bundles on his/her machine, particularly during a period of rapid functional change. In such cases, it is important to differentiate and clarify the functionality provided by each bundle.

For building all or part of an IHTSDO Workbench IDE bundle, the software development team makes use of Maven, and Maven’s project object model (POM) and plug-ins. Plug-ins define Goals and use the metadata found in the POM to perform their task. Maven is driven by the POM’s and plug-ins used within the POMS for every Maven project. A typical IHTSDO Workbench IDE bundle would comprise a set of Maven subprojects, governed by a master “bundle” project.
Bundle structure

Each Maven project has its own POM and each Maven project typically has one primary output. The rest of this section describes the Maven projects typically used to generate an IHTSDO Workbench application bundle build.

Each Maven project for the editing application bundle build follows Maven’s recommended directory structure, naming conventions for the directory structure and the naming conventions for the primary output of the project. Some of the key projects used to define an editing application bundle project are:

**IHTSDO Workbench-api project**

The IHTSDO Workbench API project contains the large set of classes that embody most of the IHTSDO Workbench Editor functionality that might need to be referenced by those developers intending to extend or enhance the editor. Source code is not open source. Contact Informatics Inc. for details. Target code is packaged under Maven as ace-api-VERSION.jar.

**foundation project**

The foundation project contains utility classes that are general in nature and shared by all the project modules. Source code is distributed under the Apache 2 open source licence and is managed via the collabnet subversion repository at https://dwfa.aceworkspace.net/. Source code has been contributed by Informatics Inc. Target code is packaged under Maven as foundation-VERSION.jar.

**core project**

This project contains the majority of basic business logic classes that control the distributed workflow capability, including business process control and scheduling and queuing classes. Source code is distributed under the Apache 2 open source licence and is managed via the collabnet subversion repository at https://dwfa.aceworkspace.net/. Source code has been contributed by Informatics Inc. Target code is packaged under Maven as core-VERSION.jar.

**vodb project**

The Versioned Object Database (VODB) project uses the java Berkeley DB framework and engine developed by Sleepycat Software, and now managed by Oracle.

**term-mojo project**

This project manages building Maven plugins that, in turn, are used to transform terminology data in preparation for importing into standard file formats used by the IHTSDO Workbench version-oriented database, and by other applications.

Setting up for Maven builds

Maven uses a declarative dependency technique and distributed repository approach to constructing applications. Application developers invoke Maven to build an application bundle based on the versions of components specified in one or more POM (pom.xml) files. IHTSDO Workbench applications rely on a set of prebuilt java class libraries some of which were described in the preceding section as projects and others of which are described briefly in . Developers need to ensure that the appropriate and compatible versions of these java class libraries are utilised in their Maven builds.

Java class library .jar files used by IHTSDO Workbench applications

<table>
<thead>
<tr>
<th>jar file</th>
<th>primary code provider</th>
<th>project and description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ace-api-1.17.jar</td>
<td>Informatics Inc.</td>
<td>IHTSDO Workbench API class library</td>
</tr>
</tbody>
</table>

Goals are what are executed to perform an action on the project. For example, the jar:jar will compile the current project and produce a JAR. Each goal exists in a plugin, and the goal name typically reflects the plugin (e.g. java:compile comes from the java plugin).
<table>
<thead>
<tr>
<th>jar file</th>
<th>primary code provider</th>
<th>project and description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activation-1.1.jar</td>
<td>Sun Microsystems</td>
<td></td>
</tr>
<tr>
<td>browser-2.1.jar</td>
<td>Sun Microsystems</td>
<td></td>
</tr>
<tr>
<td>colt-1.2.0.jar</td>
<td>CERN-European Organization for Nuclear Research</td>
<td></td>
</tr>
<tr>
<td>commons-collections-3.1.jar</td>
<td>Apache Software Foundation</td>
<td>extensive Collection classes</td>
</tr>
<tr>
<td>concurrent-1.3.4.jar</td>
<td>BEA Systems Inc. and IBM</td>
<td>Concurrency (multithreading) support</td>
</tr>
<tr>
<td>core-2.0.13.jar</td>
<td>Informatics Inc.</td>
<td>IHTSDO Workbench Distributed Workflow Architecture core library</td>
</tr>
<tr>
<td>foundation-2.0.17.jar</td>
<td>Informatics Inc.</td>
<td>IHTSDO Workbench Distributed Workflow Architecture foundation component library</td>
</tr>
<tr>
<td>icons-1.8.jar</td>
<td>Informatics Inc. and various sources</td>
<td>IHTSDO Workbench editor icon library</td>
</tr>
<tr>
<td>jcommon-1.0.0.jar</td>
<td>JFree.org</td>
<td>a collection of useful classes used by JFreeChart,</td>
</tr>
<tr>
<td>je-3.2.74.jar</td>
<td>Oracle</td>
<td>Berkeley database class library</td>
</tr>
<tr>
<td>jfreechart-1.0.0.jar</td>
<td>JFree.org</td>
<td>2D charting library for visualising business process statistics.</td>
</tr>
<tr>
<td>jsk-lib-2.1.jar</td>
<td>Apache Software Foundation and Sun Microsystems</td>
<td>Core Jini classes</td>
</tr>
<tr>
<td>jsk-platform-2.1.jar</td>
<td>Informatics Inc.</td>
<td>IHTSDO Workbench-applied Jini classes</td>
</tr>
<tr>
<td>jung-1.7.6.jar</td>
<td>JUNG contributors</td>
<td>Java Universal Network/Graph Framework - a graphing library used for displaying terminology polyhierarchies</td>
</tr>
<tr>
<td>lucene-core-2.3.2.jar</td>
<td>Apache Software Foundation</td>
<td>Lucene Java - high performance search engine library</td>
</tr>
<tr>
<td>mahalo-2.1.jar</td>
<td>Sun Microsystems</td>
<td>Mahalo - A Jini(TM) Transaction Manager Service</td>
</tr>
<tr>
<td>mail-1.4.jar</td>
<td>Sun Microsystems</td>
<td>SMTP, IMAP and POP3 e-mail classes</td>
</tr>
<tr>
<td>reggie-2.1.jar</td>
<td>Sun Microsystems</td>
<td>Service Registry classes for Jini</td>
</tr>
</tbody>
</table>
### Installation readiness with Install4j

Although the IHTSDO Toolkit is platform independent, built on Sun Microsystems Java technologies, there are sufficient variations in the operating systems of end user machines that some tailoring of the installation of IHTSDO Workbench IDEs or viewers for common operating systems will most likely be required. The recommended product for building an installer is Install4j. Install4j has a number of features that are likely to be attractive to terminology development organisations, including:-

**Native launcher generation**

Install4j generates application launchers that are native on every supported platform: Windows executables are compiled with exe4j technology; Unix shell scripts are created for Unix platforms and application bundles are created for Mac OS X. These launchers offer flexible JRE-detection.

**Advanced JRE bundling**

One of the core requirements for a Java-aware installer builder is the ability to bundle a JRE with the installer. Install4j supports JRE download-on demand, Pack200 compression or shared installations of JRE bundles. Pre-packaged JRE bundles can be downloaded from within the install4j IDE.

**Extensive i18n support**

Install4j fully supports localizing your installer to multiple languages, offering a large number of installer languages. Single- or multi-language installers that detect the actual language at runtime can be built.

### Distribution and update via Continuum

Continuum is an Apache product for managing automatic updates and synchronisation of deployments in a distributed environment.

---

23 e.g. script files, component naming, desktop windowing environments, folder or directory locations, etc.

24 see [http://www.ej-technologies.com](http://www.ej-technologies.com) for product details
Maven is extensible through the use of user-written plugins. These are a set of java classes which extend the Maven-provided base classes to provide specific build features. The Maven plugins comprise one or more “Mojo” s (the “black magic” or code that provides the plugin capability). Even most of Maven's standard functionality such as compiling java source or producing a jar file is provided by plug-ins available from the primary Maven repository (http://repo1.maven.org/).
DWFA plug-ins from Informatics Inc. provide support for terminology data transforms, process reports and graphs.

<table>
<thead>
<tr>
<th>mojo</th>
<th>phase</th>
<th>goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>DwfaSetLog</td>
<td></td>
<td>dwfa-set-logs</td>
</tr>
<tr>
<td><strong>specifications to apply to the specified logs. If you want to specify the root log, use the logger name of “root”.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExecuteMain</td>
<td>process-resources</td>
<td>execute-main</td>
</tr>
<tr>
<td><strong>executes sql commands to generate a database or perform other such tasks</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExportAnnotatedBeans</td>
<td></td>
<td>export-beans</td>
</tr>
<tr>
<td><strong>Goal which writes tasks as java beans for the builder application. This goal exports beans using the same annotations as the export-annotated-beans goal, but relies on standard maven class loader, instead of a custom class loader that is installed on top of maven class loader. Since this goal does not rely on a custom class loader, it can automatically manage the transitive dependencies, and they do not have to be declared in the project dependency section. However, the primary dependencies must be declared as part of the dwfa-mojo plugin entry so that the maven class loader can load the dependencies prior to efforts to export the beans.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExtractAndProcessFiles</td>
<td></td>
<td>process-config</td>
</tr>
<tr>
<td><strong>Goal which writes configuration files to the output directory.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GenerateScripts</td>
<td></td>
<td>generate-scripts</td>
</tr>
<tr>
<td><strong>Goal which generates shell scripts to start the dwfa bundle.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Join</td>
<td>generate-resources</td>
<td>join</td>
</tr>
<tr>
<td><strong>joins two data files with a given key. The first file is primary, in that it is read in and establishes the hash map against which the second file is joined.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProcessProjectDirectories</td>
<td></td>
<td>process-project-dirs</td>
</tr>
<tr>
<td><strong>writes configuration files to the output directory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transform</td>
<td>generate-resources</td>
<td>transform</td>
</tr>
<tr>
<td><strong>transforms source files and puts them in generated resources.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WriteBeans</td>
<td></td>
<td>write-beans</td>
</tr>
<tr>
<td><strong>writes tasks as java beans for the builder application.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WriteConfigFile</td>
<td></td>
<td>write-config</td>
</tr>
<tr>
<td><strong>writes a default startup config file.</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WriteDirectories</td>
<td></td>
<td>write-directories</td>
</tr>
</tbody>
</table>
writes configuration files to the output directory.
Using the IHTSDO Workspace on Collabnet

Collabnet is a web-based repository and framework for collaborative project development. Informatics Inc. have established (and manage) a workspace for e-health projects that use the IHTSDO Workbench IDE and the distributed workflow environment enabled by the IDE. Development Organisations may choose to create and manage a project group within the IHTSDO Workspace, that gives its software developers access to a central source code repository (based on subversion). It also gives both the developers and terminology modellers access to wiki-style project workspaces where project information can be managed, including issue management under various life-cycle phases. The key features and advantages of using Collabnet, both for IHTSDO Workbench terminology development and for extending the IHTSDO Tookit components themselves, are discussed in About the IHTSDO Workspace.
Using the IHTSDO Workspace on Collabnet

An overview of the IHTSDO Workspace can be seen in

Most of the Maven projects described in *Maven Plug-ins* have a corresponding “project” on the IHTSDO Workbench Collabnet workspace at http://project.aceworkspace.net. Access to each project is controlled on a per project basis. Every user must be registered and can be assigned to 1 or more roles. A key role relevant to developers is the developer role. Projects can be nested, but each subproject must have a globally unique name within the Aceworkspace namespace.

Initial versions of the IHTSDO Workspace were based on the Collabnet Subversion product. In September 2008, Aceworkspace migrated to the more powerful Collabnet Sourceforge product. Further details of the functionality provided by Collabnet may be found at http://www.collabnet.net.
Data Import

Terminology Set imports to Versioned Database

Different user requirements will determine what versions of what terminologies need to be preloaded into the inbuilt database. The Workbench supports the importing of individual terminology sets into the versioned database.
Terminology Set imports to Versioned Database

The IHTSDO Workbench version-oriented database can import uuid-based data from tab-delimited text files using the Maven berkley-vodb-dir goal. The following tables describe the files that can be imported, together with their required data formats.

**concepts.txt**
A required file containing all the concepts of the terminology database, and their status and version information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept uuid</td>
<td>The <em>Universally Unique Identifier</em> of the concept. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</td>
</tr>
<tr>
<td>status uuid</td>
<td>The <em>Universally Unique Identifier</em> of the status concept associated with this record. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</td>
</tr>
<tr>
<td>primitive</td>
<td>A boolean value that determines if the concept is primitive or defined. The value is represented by a string 0 (false == defined) or 1 (true == primitive)</td>
</tr>
<tr>
<td>effective date</td>
<td>The date this change became, or will become, effective, with a required granularity of seconds. The value is represented by a string of one of the following <em>formats</em>: yyyy-MM-dd HH:mm:ss yyyyMMdd HH:mm:ss</td>
</tr>
<tr>
<td>path uuid</td>
<td>The <em>Universally Unique Identifier</em> of the development path upon which this record is committed. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</td>
</tr>
</tbody>
</table>

**descriptions.txt**
A required file containing all the descriptions of the terminology database, and their status and version information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description uuid</td>
<td>The <em>Universally Unique Identifier</em> of the description. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</td>
</tr>
<tr>
<td>status uuid</td>
<td>The <em>Universally Unique Identifier</em> of the description's status concept. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</td>
</tr>
</tbody>
</table>
### Concept UUID

The Universally Unique Identifier of the description's concept.
The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000

### Term

The description's term, represented as a string, up to 32,000 characters, and can contain XML, XHTML, or HTML markup.

Currently, only HTML or XHTML will be rendered according to the Java html rendering capabilities. Future versions of the IHTSDO Workbench editor may support the association of an XSLT style sheet with a description type, allowing dynamic rendering of XML content.

### Capitalization Status

A boolean value that represents the capitalization status of the term. The value is represented by a string 0 (false == defined) or 1 (true == primitive). The semantics of this value is specific to the terminology system being imported. Future version of the IHTSDO Workbench environment may replace this value with a concept enumeration.

### Description Type UUID

The Universally Unique Identifier of the description's type.
The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000

### Language Code

The description's language code, represented as a string. SNOMED typically assumes these language codes are ISO 639-1 codes, but any string representation is accepted by the IHTSDO Workbench Editor. Future version of the IHTSDO Workbench environment may replace this value with a concept enumeration.

### Effective Date

The date this change became, or will become, effective, with a required granularity of seconds. The value is represented by a string of one of the following formats:

- yyyy-MM-dd HH:mm:ss
- yyyyMMdd HH:mm:ss

### Path UUID

The Universally Unique Identifier of the development path upon which this record is committed.
The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000

---

### relationships.txt

A required file containing all the relationships of the terminology database, and their status and version information.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relationship uuid</td>
<td>The Universally Unique Identifier of the relationship.</td>
</tr>
<tr>
<td>status uuid</td>
<td>The Universally Unique Identifier of the relationship's status.</td>
</tr>
</tbody>
</table>
The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000

<table>
<thead>
<tr>
<th>Source Concept UUID</th>
<th>The <strong>Universally Unique Identifier</strong> of the relationship's source concept. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship Type UUID</td>
<td>The <strong>Universally Unique Identifier</strong> of the relationship's type. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</td>
</tr>
<tr>
<td>Destination Concept UUID</td>
<td>The <strong>Universally Unique Identifier</strong> of the relationship's destination concept. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</td>
</tr>
<tr>
<td>Characteristic Type UUID</td>
<td>The <strong>Universally Unique Identifier</strong> of the relationship's characteristic type concept. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</td>
</tr>
<tr>
<td>Refinability UUID</td>
<td>The <strong>Universally Unique Identifier</strong> of the relationship's refinability concept. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</td>
</tr>
<tr>
<td>Relationship Group</td>
<td>An integer, represented as a string, that designates which relationship group this relationship is part of. A designation of 0 indicates that this relationship is not part of a designated relationship group.</td>
</tr>
<tr>
<td>Effective Date</td>
<td>The date this change became, or will become, effective, with a required granularity of seconds. The value is represented by a string of one of the following formats: yyyy-MM-dd HH:mm:ss yyyyMMdd HH:mm:ss</td>
</tr>
<tr>
<td>Path UUID</td>
<td>The <strong>Universally Unique Identifier</strong> of the development path upon which this record is committed. The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000</td>
</tr>
</tbody>
</table>

**ids.txt**

An optional file that provides history information about identifiers and also allows representation of alternate UUIDs for a component, as well as alternate representation forms (snomed ids, loinc ids, oids) for identifiers.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Name</td>
<td>Data Description</td>
</tr>
</tbody>
</table>
primary uuid

The Universally Unique Identifier primary uuid of a component.

The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000

source system uuid

The Universally Unique Identifier that designates the source system (SNOMED ID, UUID, OID) of the identifier. The identifier source must come from one of the concepts in the Architectonic Concept->identifier source, otherwise the native data type will simply be represented as a string.

The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000

source id

A string that represents the identifier. This string may be an oid, UUID, SNOMED Id, LOINC id, or any other representation as determined by the source. Depending upon the type of identifier, this id may be converted into a native type internally, or be retained as a String internally. The environment will use the source uuid field to determine what native data type will be used to represent the identifier.

status uuid

The Universally Unique Identifier of the status concept associated with this identifier record.

The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000

effective date

The date this change became, or will become, effective, with a required granularity of seconds. The value is represented by a string of one of the following formats:

- yyyy-MM-dd HH:mm:ss
- yyyyMdd HH:mm:ss

path uuid

The Universally Unique Identifier of the development path upon which this record is committed.

The UUID is represented in its canonical form, consisting of 32 hexadecimal digits, displayed in 5 groups separated by hyphens, for a total of 36 characters. For example: 550e8400-e29b-41d4-a716-446655440000

illicit_words.txt

An optional file consisting of a single column, containing records of words that are not permitted for use in descriptions. The illicit words list is available through the IHTSDO Workbench api, and can be used to provide algorithmic quality assurance of content that is filtered through a workflow process.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>illicit word</td>
<td>A word not permitted for use in descriptions</td>
</tr>
</tbody>
</table>

licit_words.txt

An optional file consisting of a single column, containing records of words that are acceptable for use in descriptions. The licit words list is available through the IHTSDO Workbench api, and can be used to provide algorithmic quality assurance of content that is filtered through a workflow process.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>licit word</td>
<td>A word acceptable for use in descriptions</td>
</tr>
</tbody>
</table>
Toolkit Support and Troubleshooting

*How to determine the SNOMED code of a concept or term?*
*How to determine the version of a term or a terminology being edited?*
*How to determine the version of the IHTSDO Workbench Toolkit being used?*
*How to chain together tasks in a business process/workflow?*
*How to extend the IHTSDO Workbench Editor menu?*
*How to set user profiles?*
*How to overcome memory issues?*
*How to recover from a corrupt profile?*

The answers to a number of frequently asked questions are included here.
How to determine the SNOMED code of a concept or term?

By default, identifiers for concepts and terms, are not normally displayed in the IHTSDO Workbench Editor. Depending on the term and the terminology development methodology being used, codes could be either SNOMED SCTIDs or UUIDs. The display of identifiers can be enabled using the Identifiers Toggle described in ID Toggle ID Toggle toggleidentifiers.

How to determine the version of a term or a terminology being edited?

The IHTSDO Workbench Editor can support multiple version of multiple terminologies simultaneously. Each user can select which combination of versions and terminologies should be displayed. This is done via the preferences panel as illustrated in Setting Preferences.

How to determine the version of the IHTSDO Workbench Toolkit being used?

The IHTSDO Toolkit is not versioned as a single entity. However, each component generally has some version identifier. For source code, subversion provides for automatic versioning (and tagging of branches). For built components, predominantly jar files, Maven pom.xml files and the Maven build process are used to manage versions. The component jar files (e.g. vodb-1.20.jar) are normally found in the lib folder of a distributed application bundle.

How to chain together tasks in a business process/workflow?

The IHTSDO Workbench Editor contains a set of tasks, each representing an atomic step in a business process, such as “import a list of terms”, or “stop a businesss process instance”. The Process Builder is the component of the IHTSDO Workbench Editor that can be used to graphically combine a set of tasks or steps into a single business process template. The business process (template) can be stored, edited, loaded, executed with the Editor environment. A process can be invoked by a user through a number of different mechanisms. See Automating Business Processes.

How to extend the IHTSDO Workbench Editor menu?

By placing a business process plugin file (.bp extension) in the appropriate folder of an editor bundle, the process will automatically appear under the Menu tree of the Editor. The appropriate folder is plugins/menu.

How to set user profiles?

User profiles are established by the Software Developer. A user can modify various configuration settings via the preferences toggle in the top far right corner of the editor. A user can save a profile at any time via the Editor's File menu, but otherwise will be prompted to save her/his profile on exiting the editor.

How to overcome memory issues?
Some users may require special configuration such as changing default memory settings, which otherwise might lead to application errors such as:

Error occurred during initialization of VM
Could not reserve enough space for object heap

The IHTSDO Workbench editor is invoked from an operating system specific startup script (startAce.bat startAceLinux.sh startAceOsX.sh) which in turn, invokes the JVM with initial memory parameters, specified with the -Xms and -Xmx switches The IHTSDO Workbench application bundle has a directory config, where various configurationspecific files are located. The file ace.config specifies, amongst other parameters, a cache-size parameter. The Berkeley database implementation from Oracle (formerly Sleepycat Software) uses a configuration file je.properties, which has maxMemory or alternative maxMemoryPercent parameter which may require localisation.

How to recover from a corrupt profile?

If you cannot open a database because the profile is corrupt, here is the sequence of steps to create a new default bootstrap.ace file:

1. Delete the bootstrap.ace file in the bundle directory.
2. Start up the bundle using the provided script
3. On startup, the environment will prompt you to with a config file dialog, with a place to enter a username and a password:

   ![config file dialog](image)

Enter a new username/password such as user "bootstrap" with password of "visit.bend"
4. The environment will now create the new default profile and allow you to log in.

Note that the edit and view paths are not set up, so you may have to edit your preferences pane before expected content will show up in the taxonomy view.