SEER*Abs

System Administration Reference

May 15, 2009

Section 1: Installing SEER*Abs

The registry's first installation of SEER*Abs should be on the workstation of the registry's SEER*Abs system administrator. The system administrator will configure SEER*Abs and create a registry-specific installation file that will be distributed to other users at the registry.

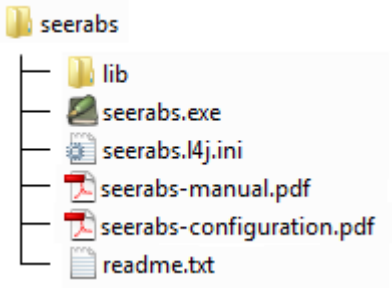
*To install and configure SEER*Abs:*

1. The SEER*Abs software requires Java 1.6.0 or later. Follow these steps to determine the version of Java installed on your computer:

    a. Enter java -version at a DOS prompt.

    b. If you do not have the required version, uninstall the existing version of Java and install the latest version. More information related to the appropriate version of Java is provided on the SEER*Abs Web site (seer.cancer.gov/seerdms/portal/seerabs).

2. Install SEER*Abs by extracting the distribution zip file into a new folder (for example, c:\seerabs). Sub-folders within seerabs.zip must be retained. To achieve this you may need to set an option in your decompression software (in WinZip, "use folder names" must be checked). Once the distribution file is unzipped, the newly created folder should contain the following files and directory:



3. Start the SEER*Abs program.

    a. Either browse to the installation folder and double-click *seerabs.exe*

    b. Or use the Windows Run command to execute SEER*Abs.

4. Define the password for the *admin* user account.

    a. Enter a value in **Password**. Passwords must contain at least 1 lower-case letter, 1 upper-case letter, and either a digit or a special character. Verify the new password by re-entering it into the **Repeat Password** field.

    b. Click **Login**.

5. The system will auto-create a SEER*Abs database and implement default configuration settings. At this point, the database will be empty except that it will contain a single user account for the system administrator. SEER*Abs only supports one administrator account and its username is always *admin*.

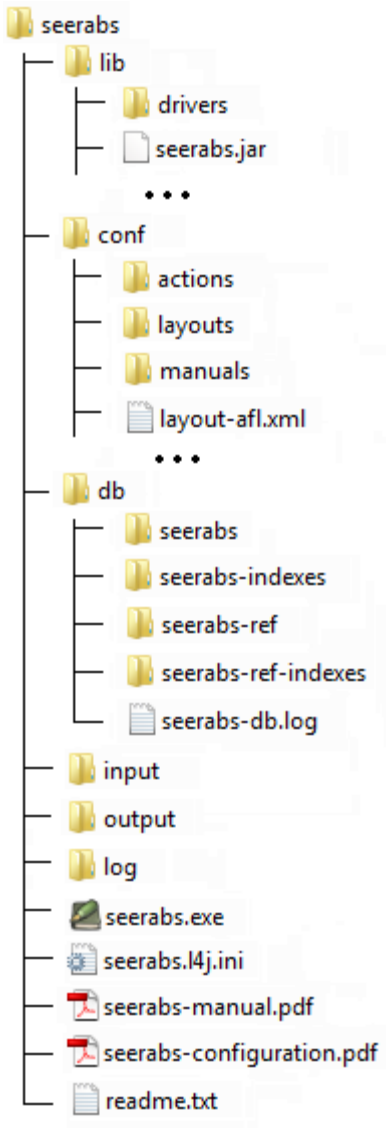
6. You are now logged in as the *admin* user. Define the record layouts and configure other system components as described in other sections of this manual.

7. Zip the configured version of SEER*Abs and install it on registry users' laptops or workstations. A separate instance of the software must be deployed for each workstation. Users cannot use the same executable or database installed on a networked drive.

8. Create a shortcut to SEER*Abs on the user's desktop or in the system tray.

Overview of SEER*Abs Files and Directories Structure

After logging in for the first time, several default directories will appear. The resulting directory structure should look like this:



The following table provides a description of each file and directory.

File / Directory	Description
seerabs	Root directory, there is no restriction about where it needs to be located on the file system when SEER*Abs is installed, except that it must be in a writable directory (since SEER*Abs will automatically create sub-directories the first time it is started).
lib	Library directory, it contains all the libraries (jar files) required by the application. Updating the application will involve updating some of the jar files, added some new ones and deleting some old ones.
lib / drivers	Sub-directory in the library directory. It contains the jar files necessary for SEER*Abs to communicate with other databases (drivers). Those drivers libraries just need to be dropped in this directory; SEER*Abs (after being restarted) will parse the directory and load any drivers present in it. Once loaded, the drivers will enable the synchronization scripts to communicate with a remote database. Only one set of drivers is shipped with the SEE*Abs release: ojdbc6.jar; that drivers file enables SEER*Abs to communicate with Oracle databases.
conf	Configuration directory. It contains any files related to the configuration of SEER*Abs. Copying “a configuration” from one SEER*Abs installation to another one means copying this entire directory and overriding the target configuration directory with it. The files contained in this directory are expected by the application; they should not be renamed. While files from this directory can be modified through some external programs, it is recommended to use the build-in file editor, which contains validation features. If a file is corrupted by modifying it outside of the application, SEER*Abs will fail the login process.
conf / actions	Sub-directory of the configuration directory that contains user-defined scripts (called actions in SEER*Abs). Once defined, those actions are available through the “Action” menu. One special file in that directory is the “actions-info.data” file. It contains information about each user-defined script and which label should be used in the “Action” menu. The file should never be modified as SEE*Abs uses it internally and a corrupted file might prevent the application from starting.
conf / layouts	Sub-directory of the configuration directory that contains user-defined input layouts. Once defined, those layouts can be used in scripts to request user input.

File/Directory	Description
conf/manuals	Sub-directory of the configuration directory that contains manuals. One special file in that directory is the "manuals-info.data" file. It contains information about each manual file and which label should be used in the "Help" menu. The file should never be modified as SEE*Abs uses it internally and a corrupted file might prevent the application from starting.
db	Database directory. SEER*Abs uses the Derby Java database (http://db.apache.org/derby/). SEER*Abs uses two types of data: regular data and reference data. The first set of data is extremely important since it corresponds to work done by abstractors. The second set of data is far less important since it is read-only data that can always be recovered by fetching it again from the Central Registry. For that reason, the SEER*Abs database has been split into two databases: a main database and a reference database. Those correspond to the sub-directories described hereafter. Derby is a file-based database. It is therefore technically possible to move an entire database (main or reference) from one SEER*Abs installation to another one by copying the appropriate sub-directory. When doing so, it is important to copy the corresponding indexes sub-directory or the searches might not return correct results in the target installation. If the indexes are not available, then at the very least they should be deleted from the target installation in which case SEER*Abs will automatically re-create them when starting up.
db/seerabs	Main database. Contains data that cannot be lost. SEER*Abs provides a backup mechanism that should be used frequently. It also provides a corresponding restore mechanism.
db/seerabs-indexes	Main database indexes. This directory should never be apart from the main database directory. The indexes are created by a search library called Lucene (http://lucene.apache.org/java/docs/) and should never be modified manually.
db/seerabs-ref	Reference database. Contains data that can be recovered by synchronizing SEER*Abs with the main registry again. There is no backup/restore mechanism for this database, but SEER*Abs provide an option to copy the entire database (and its index directory) from a chosen directory. This is useful if an IT person wants to download the reference data once (since it can be a very slow process) and make the reference database available to all the other SEER*Abs installations on a shared network.
db/seerabs-ref-indexes	Reference database indexes. This directory should never be apart from the reference database directory. The indexes are created by a search library called Lucene (http://lucene.apache.org/java/docs/) and should never be modified manually.
db/seerabs-db.log	Database log files. Any errors will be logged in the SEER*Abs log files, but Derby also creates its own log.

File / Directory	Description
input	Input directory. When importing files through the synchronization module, SEER*Abs will open this directory by default when requesting a location for the file(s) to import. This is only a default and can be modified by the user.
output	Output directory. When extracting files through the synchronization module, SEER*Abs will create the file in this directory by default. This is only a default and can be modified by the user.
log	Log directory. When an error happens, it is logged in the file "seerabs.log" contained in this directory. This should be the first place to look when trying to resolve bad application behavior reported by the abstractors. An exception in the log does not mean that SEER*Abs contains an error. It can also mean that the configuration has not been properly set up (most likely an error in a script).
seerabs.exe	Executable file used to start the application. On Windows, a shortcut can be created from this file by right-clicking it and selecting the option "create shortcut".
seerabs.l4j.ini	Initialization file. SEER*Abs is a Java application and therefore uses a Java Virtual Machine (JVM) to run. The parameters from this initialization file are passed to the JVM. An important parameter is the maximum memory that the JVM can use. By default it is set to 512MB. That parameter should be adjusted if SEER*Abs is used on a computer that does not have that amount of memory available. Setting that parameter to a bigger value should not affect the application at all since it should not require that much memory under any circumstances.
configuration-manual.pdf	This manual.
user-manual.pdf	The user manual.
readme.txt	Readme file with instructions to do a quick install and startup. All the information in that file is contained in this manual.

Section 2: SEER*Abs Data Types

SEER*Abs handles data using the concept of "entity". An entity is an instance of a particular type of data, for example an abstract record, or an AFL. Every entity has a type associated with it; those types cannot be customized and are described in the following table.

Entity Type	Type name to use in Scripts	Database	Description
AFL	AFL	main	Abstract Facility Lead. Represents a case that needs to be investigated in a facility and maybe abstracted into a record. AFL cannot be created in SEER*Abs and need to be imported.
Record	RECORD	main	Records created in SEER*Abs like abstracts or casefindings. SEER*Abs can also be configured to support other record types if needed. Records cannot be imported.
User	USER	main	User logging in SEER*Abs and using it. By default, the application supports a unique user which has administrator privileges; its username is 'admin' and it is the only administrator user that is created. Any number of non-administrator users can be added by the admin user. Users cannot be imported; they need to be created within the application.
Facility	FACILITY	reference	Represents a facility (hospital, lab, etc…). Such a facility can be provided during the login process to apply a default filter to the worklist. Facilities cannot be modified or created in SEER*Abs and need to be imported.
Physician	PHYSICIAN	reference	Represents a physician. Only used when creating new records. Physicians cannot be created in SEER*Abs and need to be imported.
Reference Record	REFERENCE-RECORD	reference	Records used as reference data only. They cannot be created or modified through SEER*Abs and need to be imported.
Reference Patient	REFERENCE-PATIENT	reference	Patient Sets used as reference data only. They cannot be created or modified through SEER*Abs and need to be imported.
Lookup	N/A *	reference	Lookup, used to format a code into a label.

** Lookups are handled a little bit differently than the other entity types since they don't have customizable properties. For that reason specific methods have been added for them in the script utility methods (see Annex 1).*

The second column provides the string that needs to be used when referencing a particular type in a script (many utility methods require a type as a parameter). The third column provides which database is used when persisting the entities of that type; if using the main database then all those entities will be saved when doing a backup of the application's data; the entities using the reference database won't be saved.

Subtypes

While it is true that the types are not customizable, some of them have a subtype which is customizable. It is true for the Record and Reference Record types. Which subtype they support is defined in the main configuration file. For the records, the following property is used:

```
supported.record.subtypes=abstract,casefinding
```

And for the reference records, the following is used:

```
supported.ref.record.subtypes=naaccr,casefinding,hl7
```

Those lists of subtypes can be modified; there is no restriction on the values of the reference record list, but "abstract" and "casefinding" are both required in the record list. Also note that if a new record subtype is defined, a corresponding prefix also needs to be provided (see Main Configuration section of this manual for more details).

Most of the utility methods dealing with entities require both a type and a subtype. For types that do not have a subtype, null should be passed as a parameter. For example, the following call saves a list of AFLs:

```
env.utils.saveEntities("AFL", null, aflsToSave)
```

While the following call saves a list of abstracts:

```
env.utils.saveEntities("RECORD", "abstract", recordsToSave)
```

See the *Utility Methods* annex for more details about the available methods and how to use them.

Defining New Properties

Lookups are a special type of entity and are defined in their own configuration file (see *Defining Lookups* section). All the other entity types are defined in a Layout. That layout contains the properties that should be shown on the screen along with some other information (property type, label, lookup, etc…). While it is true that the layout is mainly used to define where the fields should be shown on the screen, it is also used to define which properties are supported for which entity type. An entity can be seen as a map of keys and values. The keys are the field names defined in the layout and the values are the text corresponding to those field names (it can be the text typed by the abstractor in the editor, or the text downloaded through the synchronization module). For efficiency, a key corresponding to an empty (null) value is not saved in the database; that means the absence of a key in a map should be interpreted as the key having an empty value. That also means different entities of the same type will end up with different keys, depending

which values are missing. For that reason, there are no database constraints linking the properties to the entity types; saving an entity in the database means saving a generic mapping of keys and values; the database is unaware of which properties the mapping should have depending on the entity's layout.

With this design, adding a new property to a given type is as simple as adding a field to the corresponding layout. Once added, the abstractor will be able to provide a value to that field; that value will be persisted in the database and made available to the synchronization scripts to be exported. Any properties can be defined in a layout, but a few of them are used by the application and therefore SEER*Abs needs to be aware of their name. Most of those internal properties can be re-defined in the main configuration in case they conflict with other regular properties. The following properties are currently used by SEER*Abs:

Property Name	Can be re-defined in Main Conf	Description
displayId	yes	Entity display identifier; usually a few uppercased letters followed by a dash followed by a numerical part (for example FAC-123). That display ID is what is shown on the screen to uniquely identify an entity. Applied to all the entity types.
type	yes	Entity type; correspond to the string that needs to be used in the scripts (see data types table at the beginning of this section). Used by SEER*Abs to filter the worklist, the searches, to call the correct utility methods, and for many other things. Every entity has a type.
subtype	yes	Entity subtype; correspond to the subtypes defined in the main configuration. Applies only for RECORD and REFERENCE-RECORD types.
status	yes	Entity status; used to indicate in which status an entity currently is; the statuses can be defined in the lookup configuration file. It is mainly used by SEER*Abs to filter the worklist; for that purpose the main configuration also provides a way of defining which status is an "outstanding" status. Those are filterable under the "<Outstanding>" option in the status filter. Statuses apply only to AFL and RECORD (any subtypes).
dateLastModified	yes	The last time an entity was modified. It is automatically updated by SEER*Abs every time an entity is persisted. Applies to any entity type.

Property Name	Can be re-defined in Main Conf	Description
facility	yes	Main facility for an entity. Used by SEER*Abs to filter by facility. Applies to AFL, RECORD, REFERENCE-PATIENT and REFERENCE-RECORD.
primarySite	yes	Entity site. Used by SEER*Abs to dynamically build the content of the collaborative stage lookup. Applies only to the RECORD type.
histologyICDO3	yes	Entity histology. Used by SEER*Abs to dynamically build the content of the collaborative stage lookup. Applies only to the RECORD type.
username	no	Uniquely identifies which user is currently logged in. Applies only to USER type.
password	no	Used during the login process and when an abstractor wants to change his/her password. Applies only to USER type.

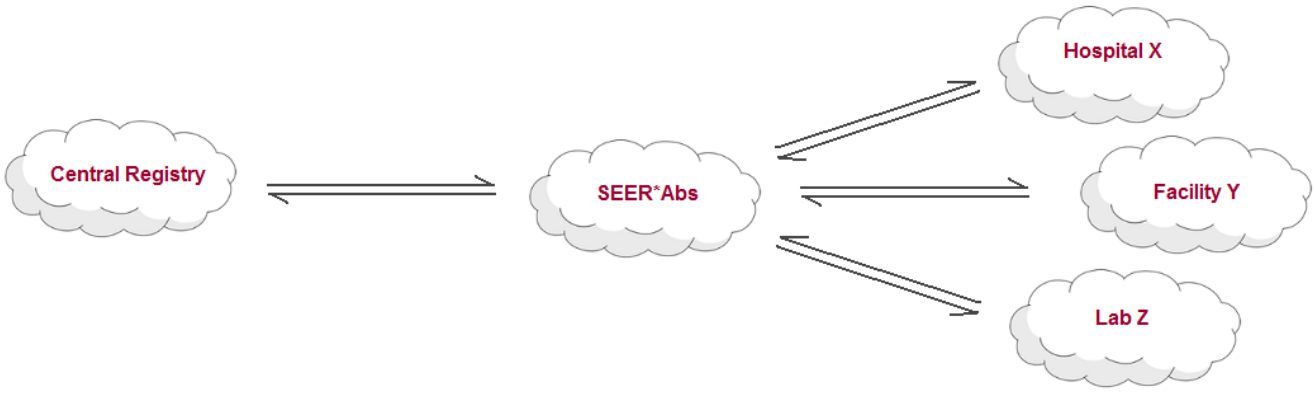
There are a few other properties used internally by SEER*Abs (like a database ID for example) but those should never be referenced by any scripts and therefore are not described here (they usually start with a double underscore).

Having to re-define an internal property in the main configuration should be extremely rare. For example if a new reference record type is added and has to use the property “dateLastModified”, the internal property with the same name could be re-defined as “dateLastModifiedSeerabs” to avoid any conflict. But the script downloading that new reference record type could also save that new “dateLastModified” property under a different name and therefore also avoid the conflict. Note that if a property is re-defined, all the data needs to be fixed (for reference data, it means deleting the old data and re-importing it; for the main data it means running an action script that would load all the entities of that type and for each of them remove the old property and re-add the new one).

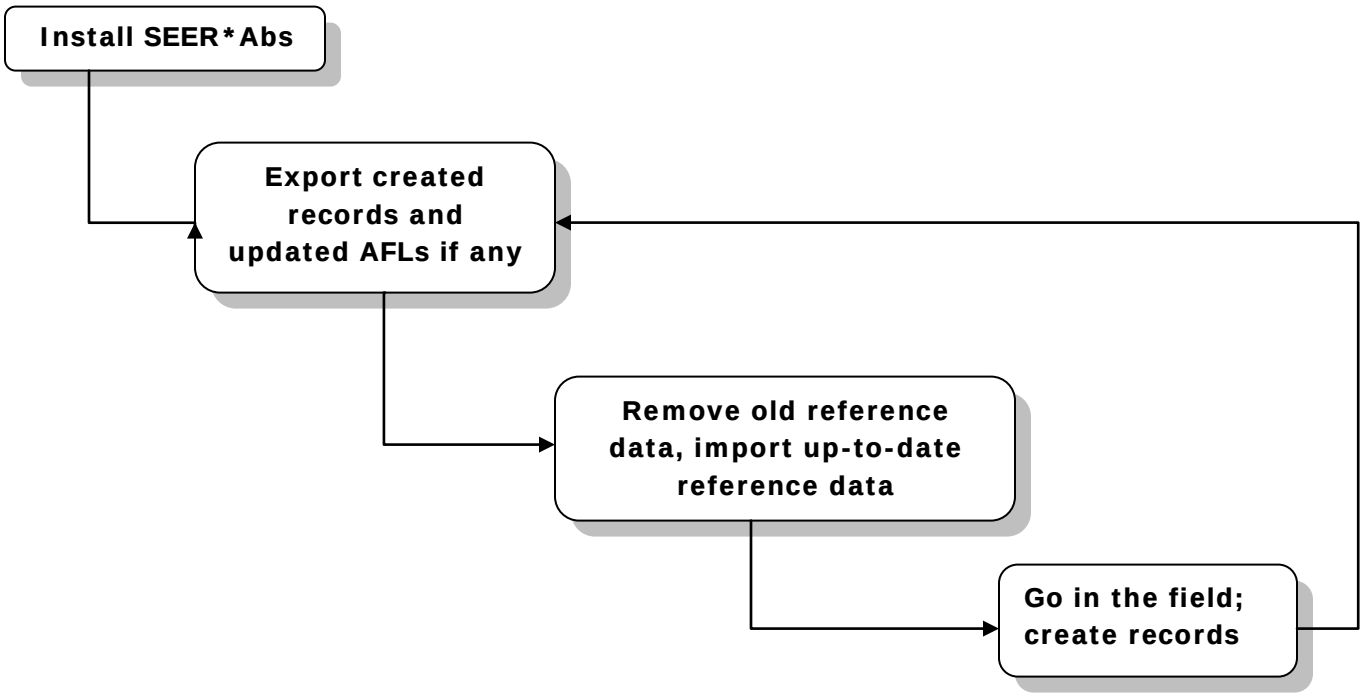
Because empty values are not saved in the database, different entities of the same type could have different properties saved in the database. For that reason, a script cannot make any assumptions on which properties is supposed to be contained in an entity. This can be annoying when trying to write scripts that reference hundreds of properties. To solve that problem, a utility method is provided; for a given type and subtype, it returns a list of properties as they are defined in the corresponding layout. See *Annex 1* for more details about utility methods.

Section 3: Defining a Workflow

Before configuring SEER*Abs, it is important to decide how it is going to be used. The following picture shows the different components playing a role in the workflow.



SEER*Abs is designed to be an Abstracting Tool that is used in the field to create Abstracts (or other record types). It then communicates with the Central Registry (through the synchronization module) to export the created records and maybe import the reference data. That default workflow is shown in the following figure:



By configuring the synchronization module, a Registry can define how the records and AFLs are exported and how the reference data is imported. By configuring the editor module, a Registry can define how the records are created (what fields, what format, etc...).

Abstractors need to know whether a record has just been created, whether all the work is done for it or whether it has already been exported. A status field is used for that purpose. Only the AFL and the RECORD entity types have that field. The possible values are defined in the lookups *lkup_internal_alf_status* and *lkup_internal_rec_status*. Those lookups cannot be deleted from the configuration but their content can be modified and that is the main mechanism to customize the SEER*Abs workflow.

The following AFL statuses are provided with the default configuration:

Status Code	Status Label	Description
1	NOT PROCESSED	No work has been performed on this AFL.
2	IN PROGRESS	Some work has been performed on this AFL
3	NOT ABSTRACTED	The work on this AFL is done; it has not been abstracted and a reason has been provided.
4	ABSTRACTED	The work on this AFL is done; an abstract has been created.
5	ARCHIVED	This AFL has been exported.

The following record statuses are provided with the default configuration:

Status Code	Status Label	Description
1	IN PROGRESS	Some work has been performed on this record.
2	COMPLETED	No more work needs to be performed on this record.
3	ARCHIVED	This record has been exported.

Note that changing the label of a status has no impact on the workflow and no script needs to be modified in that case. On the other hand, many scripts use the status code to search entities and load them (for example the extract script fetches all the records with a status of 'COMPLETED'). Adding, removing or changing the meaning of a status (how it is supposed to be used by the scripts) require to review each script and make sure the way it uses the statuses (if it does use them) is still correct.

SEER*Abs never deletes any record or AFL automatically. A special action is provided (Purge Entities) to delete any entities with a status of 'ARCHIVED'. That action can be run right before

exporting records so the ones exported from the previous synchronization session are deleted and the new ones are marked as 'ARCHIVED' after being exported. But the user has to trigger the action manually and exactly when that should happen must be a Registry policy.

To define a different workflow than the default one:

1. Log in as the administrator.
2. Open the Configuration module.
3. Double-click the unique row of the Lookups section to open the corresponding file in the Configuration File Editor.
4. Modify the *lkup_internal_afl_status* and *lkup_internal_rec_status* lookups; add the statuses that define the new workflow.
5. Once done, close the editor.
6. If prompted to save the changes, click Yes.
7. The lookups are automatically reloaded in the application.
8. Open each script and action script, search for the word 'status' and make sure the way the script uses the status (if it does) comply with the new definition. Synchronization scripts will certainly have to be updated since they heavily use the statuses.
9. Let the abstractors now what the new statuses are and what they mean.

Section 4: Configuring SEER*Abs

The SEER*Abs configuration manager can be used to customize all system features including data entry screens for records, the Search page interface, and synchronization scripts. The manager allows you to open files in the SEER*Abs configuration editor by selecting the corresponding row and typing *Enter* or by clicking on the edit icon in the table's action column. All configuration files are text files; therefore it is technically possible to edit them in any text editor. However, it is recommended that you use the SEER*Abs configuration editor to take advantage of the validation, preview, and auto-refresh features.

The configuration files are organized into the topics listed below.

- **Main Configuration** – a single configuration file containing system parameters and properties. These include system options, defaults, and global variables.
- **Layouts** – separate configuration files defining the screens displayed when you view or modify a record, AFL, patient set, facility, physician, or user account. These include the layouts used to display records created in SEER*Abs and reference patient sets and records.
- **Input Layouts** – special layouts that can be referenced in any script to request user-input.
- **Searches & Filters** – these files define the filters and tables used to display and find data on the Search page, the Worklist, and the User account manager.
- **Scripts** – Groovy scripts run to create extract files, load reference data from external files or directly from a database, and utility scripts used to create AFLs and implement edits.
- **Action Scripts** – special Groovy script that the abstractor can run through a menu item.
- **Lookups** – a single configuration file defining lookup tables for data fields. This file contains full definitions for internal lookups and reference information for external lookups.
- **Edits** – source code for edits to test the validity of data fields. Two sets of edits are integrated into SEER*Abs: edits developed by the NCI SEER Program that cover data fields submitted to SEER; and edits developed within SEER*Abs. The SEER*Abs edits may include edits shipped with the software as well as the edits created by your registry staff.
- **Autocomplete Terms** – the list of words and phrases matched against the user's entry when the autocomplete feature is used.

Each of those topics is explained in details in the following sections.

SEER*Abs will work out-of-the-box without configuring anything. But it is recommended to review the entire configuration the first time the application is installed to make sure that the default configuration fits with your Registry's operations. The checklist provided in the next paragraph can be used as a reference to verify the entire configuration.

Checklist for Reviewing the SEER*Abs Default Configuration

1. Start with the main configuration file, the **registry.id** property should be set to the correct SEER Registry ID; if the application needs to connect to SEER*DMS (or another external database), the **synch.db.connection.1.url** property need to be modify to point to that external database. The corresponding **synch.db.connection.1.username** property can also be set to a default username so the abstractor does not have to always re-type it. Review the list of record types that the application needs to create, this is the **supported.record.subtypes** property. Review the reference record types that need to be supported, this is the **supported.ref.record.subtypes** property. If adding, removing or modifying any subtypes, the application will need to be restarted for the changes to take effect. Once this is done, review the other properties from the main configuration; make sure you understand what they do and whether their current value suits your need.

2. Switch to the layouts, there should be one layout for each entity type except for the RECORD and REFERENCE-RECORD types which have one layout per subtype. Edit each record layout and use the preview tab to see how they look. Make sure they support the fields you need. The default configuration is shipped with two record types:

    - *abstract*: **layout-record-abstract.xml** file; implemented according to the NAACCR 11.3 documentation. If adding, removing or renaming fields in that layout, the corresponding **script-extract-record-abstract.groovy** script will need to be modified to correctly export those fields. The **script-action-record-create.groovy** script might also have to change to correctly copying fields from an existing record to a new abstract (using the copy-into-abstract feature).

    - *casefinding*: layout-record-casefinding.xml file; the default layout does not use any particular standard; the corresponding extract uses a Comma Separated Values (CSV) format. If adding, removing or renaming fields in that layout, the corresponding **script-extract-record-casefinding.groovy** script will need to be modified to correctly export those fields. The **script-action-record-create.groovy** script might also have to change to correctly copying fields from an existing record to a new abstract (using the copy-into-abstract feature).

3. Review the AFL layout (**layout-afl.xml**); edit it and use the preview tab to see how it looks like. In the script section, review the corresponding import script (**script-import-afl.groovy**); the default script deletes any UNPROCESSED AFLs and download any AFL from SEER*DMS that is opened (status is 1). Review the export script (**script-load-afl.groovy**); default script creates a mass change file for all the AFL that were not abstracted (mass change file includes the reason the AFL was not abstracted and a comment). The AFLs that were abstracted are not included because they will be closed in SEER*DMS by the incoming abstract. Once the export is completed, all the processed AFLs are marked as ARCHIVED.

4. Review all the reference data layouts and their corresponding import/export scripts:

- *Lookups*: those do not have a layout but instead are defined in their own file (**lookups.xml**). Review the file and make sure any lookup required in the application is defined there. If a lookup is defined as external, its content needs to be loaded from the import script (**script-load-lookup.groovy**), otherwise the content needs to be defined in the lookup XML file it-self.

- *Facilities* (**layout-facility-xml**): default import script (**script-load-facility.groovy**) loads all active facilities from SEER*DMS.

- *Physicians* (**layout-physician.xml**): default import script (**script-load-physician.groovy**) loads all active persons that are marked as medical practitioner from SEER*DMS.

- *Patient Sets* (**layout-ref_patient.xml**): default import script (**script-load-ref_patient.groovy**) loads all the Patient Sets that are ALIVE with a year of last contact after 2000. Patient Sets are multi-level entities (a single Patient Set contains a list of CTCs; each CTC contains a list of Facility Admission, etc…). Because of that, loading Patient Sets from SEER*DMS requires several database call per entity. This will make the synchronization process extremely slow. To speed up the process; the criteria that selects which Patient Sets to load can be made more strict, or less information per Patient Set can be loaded (for example loading only demographic information).

- *Reference Records*: the default configuration supports three types of reference records:

    - *naaccr* (**layout-ref_record-naaccr.xml**): default import script (**script-load-ref_record-naaccr.groovy**) loads all unlinked NAACCR records from SEER*DMS. Downloading NAACCR records can be a slow process since they are composed of hundreds of properties. Downloading less of them will make the synchronization process faster.

    - *casefinding* (**layout-ref_record-casefinding.xml**): default import script (**script-load-ref_record-casefinding.groovy**) loads all unlinked Casefinding records from SEER*DMS.

    - *hl7* (**layout-ref_record-hl7.xml**): default import script (**script-load-ref_record-hl7.groovy**) loads all unlinked HL7 records from SEER*DMS. Downloading HL7 records can be a slow process because of the long text they contain. Downloading less of them will make the synchronization process faster.

5. Switch to the Edits section and review the SEER*Abs edits (seerabs-edits.xml). The SEER edits are always shown in the configuration even if they are not loaded (see **edits.load.seer** and **edits.ignore** properties in the main configuration); they are provided as a reference but are read-only and can never be modified through the application. The few SEER*Abs edits that are provided in the default configuration should only be used as a reference to write more complicated ones. They are not intended to be used in production.

6. In the Users module, use the *Add User* button to add new users. The users can be managed in two ways (see *Managing Users Accounts* section for more details):

    - Do not add any users in the initial configuration; instead install SEER*Abs on each laptop with no regular users (there will always be an admin user). Then add a single (different) user on each laptop.

    - Add all the users during the initial configuration. Copy all the users on each laptop (remember that the users are saved in the main database; they are not part of the configuration; therefore the main database along with its index directory will need to be copied over to the laptops).

    The user layout (**layout-user.xml**) can be used to add new properties to the USER entity. For example the abstractor ID has been added to the default configuration since it is a NAACCR field and is used to automatically fill-in the corresponding field when creating new abstracts.

Section 5: Main Configuration

Global configuration parameters are set in the Main Configuration file (**seerabs.properties**). All of these parameters must be included in the file unless they are designated as optional below.

Review and adjust these parameters before configuring the layouts and scripts. Changes to some parameters are applied when you save and close the configuration editor. Other changes are not applied until you close and restart SEER*Abs, as prompted. Any changes made via a text editor will only be applied when you restart SEER*Abs.

Property	Default Value	Description
editor.enable.auto.forward	true	Whether or not the auto-forwarding mechanism should be enabled in the editor (true or false). If true, the cursor in the editor will automatically go to the next field when the number of characters in the current field equals the field's length. If false, the user will always need to press tab to advance to the next field.
editor.enable.auto.validate	true	Whether or not the auto-validating mechanism should be enabled in the editor (true or false). If enabled, validation (edits evaluation) will happen every time the abstractor tabs out of a field. If false it will only happen when the record is open/saved/closed. This parameter has no effect on internal edits.
editor.enforce.lookup.validation	true	Whether a non-blank value that is not in the corresponding lookup should generate an internal edit (true or false). This parameter has no effect on fields that do not have a lookup or on special collaborative stage lookups.

Property	Default Value	Description
editor.internal.edits.strict.validation	true	Whether the abstractor should be warned when an internal edit is triggered (true of false). An edit error will always be displayed in red, but if this parameter is set to true, if a wrong value is entered (an alpha characters in a numeric field, a value that is not in the corresponding lookup, etc...), there will be a beep and the focus will stay on the current text field with the value highlighted (so it can be easily replaced).
edits.ignore		Individual edits to ignore (comma-separated list of edit ID's). Only edits that are loaded can be ignored (so if the SEER edits are not loaded, only SEER*Abs edits can be in this list).
edits.load.seer	false	Whether or not the SEER edits should be loaded (true or false). If set to true, the validation of an abstract record will include the SEER edits. If set to true, the *edits.ignore* property should be reviewed.
import.ref.db.default.path		When importing an entire reference database, this default path will be used as a default directory when the directory selection dialog is displayed. If this property is left blank, the OS default user directory will be used.
property.date.last.modified	dateLastModified	Property to use as Date Last Modified.
property.display.id	displayed	Property to use as Display ID.
property.facility	facility	Property to use as Facility.
property.histology	histologyICDO3	Property to use as Histology.
property.site	primarySite	Property to use as Site.
property.status	status	Property to use as Status.
property.subtype	subtype	Property to use as Subtype.
property.type	type	Property to use as Type.

Property	Default Value	Description
record.prefix.abstract	ABS-	Display ID prefix to use when creating new abstracts.
record.prefix.casefinding	CF-	Display ID prefix to use when creating new casefinding.
registry.id	0000000000	SEER registry ID.
supported.record.subtypes	abstract, casefinding	Which record types can be created from SEER*Abs. When defining a new record type in this list, the application will automatically create an empty layout and some empty scripts that can be modified through the configuration manager. The entries defined in the list are used as file names, only alpha-numeric characters and/or underscore (_) can be used. The two first entries ('abstract' and 'casefinding') are required by the application and cannot be removed or renamed. The provided entries should also be defined in 'lkup_internal_rec_types' in the lookup XML file so the application displays a nice name instead of the ID defined here. So for example, if one wants to add support for short abstract, one would set the list to 'abstract,casefinding,short_abstract', add 'short_abstract -> Short Abstract' to lkup_internal_rec_types and modify any relevant layout and/or scripts for that new type.
supported.ref.record.subtypes	naaccr, casefinding, hl7	Supported record types in the reference data. The entries defined in the list are used as file names, only alpha-numeric characters and/or underscore (_) can be used. The provided entries should also be defined in 'lkup_internal_reference_rec_types' in the lookup XML file so the application displays the nice name instead of the ID defined here.

Property	Default Value	Description
synch.db.connection.1.name	SEERDMS	A value must be set for this parameter if *synch.methods.enabled* includes *import-database*. This property identifies a database connection to the abstractors. This text is shown in the drop-down box when the user starts importing reference data using the Load from Registry Database method. You may specify up to nine connection names by adding parameters to the main configuration file (synch.db.connection.1.name to synch.db.connection.9.name). There must be a URL defined for each connection name. The connection names must be unique.
synch.db.connection.1.url	jdbc:oracle:thin:@your.server.address:1521:sid	A value must be set for this parameter if *synch.methods.enabled* includes *import-database*. This is the connection URL for the database defined in synch.db.connection.1.name. The URL format is specific to the database vendor (Oracle, PostGresSQL, Apache Derby, MySQL, or any other vendor). The default settings shipped with SEER*Abs are in the format required to connect to the SEER*DMS Oracle database. For SEER*DMS databases, this is the same connection string that you use in Workbench or other SQL tools: o your.server.address: you would typically use the data warehouse hostname. o sid = Server ID. Use seerdw if you wish to connect to the SEER*DMS data warehouse. You may specify up to nine connections by adding name and URL parameters to the main configuration file.

Property	Default Value	Description
synch.db.connection.1.username		A default name for the corresponding connection; can be left blank. If a username is provided, it will be used to auto-fill the username box in the dialog that request the abstractor to provide the connection username and password when synchronizing.
synch.export.default	export-file	The Method drop-down menu will use this as the default in the export section of the Synchronization module. This parameter is can be left blank, in which case SEER*Abs will pick a random default value.
synch.import.default	Import-database	The Method drop-down menu will use this as the default in the import section of the Synchronization module. This parameter is can be left blank, in which case SEER*Abs will pick a random default value.
synch.methods.enabled	export-file, import-file, import-database	This comma-separate list determines the methods that the abstractors will be able to use to synchronize SEER*Abs with the registry's main database. The following are available: • **export-file**: export entities by creating local files (extracts) • **export-database**: export entities by connecting to a remote database • **import-file**: import entities from local files • **import-database**: import entities from a remote database
title	SEER*Abs	A short version of the application title. It is shown in the title bar of child windows, dialogs and popups.
title.main	SEER Abstracting Tool	A longer version of the application title. This is shown in the main SEER*Abs window.

Property	Default Value	Description
worklist.oustanding.statuses.afl	1,2	What AFL status is considered as "outstanding" (comma separated list of codes defined in AFL status lookup); this is used in the worklist filter to show the outstanding work; if left blank then the outstanding option won't be available.
worklist.outstanding.statuses.record	1	What record status is considered as " outstanding " (comma separated list of codes defined in record status lookup); this is used in the worklist filter to show the outstanding work; if left blank then the outstanding option won't be available

In addition to those properties, user-defined properties can be added to the main configuration and accessed by the script through a utility method. For example, the following line could be added in the main configuration:

```
laptop.number=15
```

Then in the script that runs when creating new abstracts, the following call could be made to get the value of that property from the configuration and automatically assign it to a field:

```
env.utils.getConfVariable(“laptop.number”)
```

This is useful to define a variable that needs to be different for each laptop installation without having to modify all the scripts.

Section 6: Defining Layouts

XML configuration files define the display screens for record, AFL, patient set, facility, physician, and user account data. There is a separate XML file for each of the following:

- AFL page
- Data Entry screen for each record type defined in the main configuration
- Screens to view reference data:
    - Facility
    - Physician
    - Any record type defined in the main configuration
- User Account page

The layout files should be edited via the SEER*Abs editor to take advantage of the validation, preview, and auto-refresh features. However, the XML files are stored in the *conf* installation folder and can be opened with any text editor. The files are named with the format "layout—type-subtype.xml" for the RECORD and REFERENCE-RECORD types and "layout-type.xml" for the other types.

To modify a layout:

10. Log in as the administrator.
11. Open the Configuration module.
12. Locate the layout that needs to be modified.
13. Double-click the row containing that layout to open it in the Configuration File Editor.
14. Modify the XML. For example, to add a field:
    a. Locate the tab, section and row in which the field should be added
    b. Add a field tag with all required attributes
    c. Click the **Validate** button to verify the XML syntax
    d. Click the **Preview** panel to review your changes
    e. Click **Close**
    f. When prompted whether the modifications should be saved, click **Yes**. The editor will close and SEER*Abs will reload the layout (making it available to the application).

XML Structure for Layouts

The following shows the available XML tags and their hierarchy in layout configuration files. The on-entity-opened, on-entity-saved, and on-entity-exited tags are optional. All other tags are required. As shown in this sample, section is the only tag that can be nested. Nested sections should be used for grouping fields that go together logically.

```
<editor-layout>
    <tab>
        <section>
            <row>
                <field>
                    <on-entity-opened>
                        Groovy script that executes when an entity is opened via this layout
                    </on-entity-opened>
                    <on-entity-saved>
                        Groovy script that executes when an entity is saved via this layout
                    </on-entity-saved>
                    <on-field-exited>
                        Groovy script that executes when the field loses focus
                    </on-field-exited>
                    <description/>
                </field>
               <section/>
            </row>
        </section>
    </tab>
</editor-layout>
```

editor-layout tag

This is the root XML tag for a SEER*Abs layout. This tag is required and can only be defined once in a layout file. The editor-layout tag has the following attributes:

- desc (optional): short description for this layout, will be shown in the configuration module.
- default-case (optional): If this attribute's value is upper or lower, the case of each field will be converted unless a case attribute is specified in the field tag. Valid values are 'upper', 'lower' and 'as-is'. If you do not specify a value, the default case will be 'upper'.

- default-trim (optional): This rule will be used to remove leading and trailing white space from each field unless the trim attribute is specified in the field tag. Trim may have the following values: “right”, “left”, “both” and “none”. If a value is not specified, “right” is used by default.

- default-gap-before-label (optional): The number of pixels of white space displayed to the left of each field label. 10 pixels are used if this attribute is not specified. You may over-ride this attribute by setting the *gap-before-label* attribute for individual fields.

- default-gap-after-label (optional): The number of pixels of white space displayed to the right of each field label. 3 pixels are used if this attribute is not specified. You may over-ride this attribute by setting the *gap-after-label* attribute for individual fields.

- default-gap-before-row, default-gap-after-row (optional): The number of pixels of white space displayed above each row. If not specified, 5 pixels are used by default. You may over-ride these attributes by setting the *gap-before-row* for individual rows.

- default-gap-after-row, default-gap-after-row (optional): The number of pixels of white space displayed under each row. If not specified, 5 pixels are used by default. You may over-ride these attributes by setting the *gap-after-row* for individual rows.

tab

SEER*Abs layouts support multiple tabs or pages to display content. You must define at least one tab tag. The tab tag has the following attributes:

- label (required): This label is displayed on the page’s tab control when there are multiple tabs. If there is only one tab in a layout, the tab control and label are not displayed.

- repeating (optional, only applicable to multi-level entities; only the patient set is a multi-level entity in the default configuration): This attribute identifies a sub-entity that can have multiple instances within the layout’s main entity. This attribute defines data structure and display. For example, you could define “repeating=ctcs” at the tab or section level in the Patient Set layout (the samples shipped with SEER*Abs actually have this attribute at the section level). Because “ctcs” is defined as repeating, data for multiple cancers can be loaded for each patient set. The path in the import and load scripts is the value of this attribute (“ctcs”). Defining the attribute at the tab level means that the repeating ctcs will be shown on separate tabs of the interface. Repeating attributes can also be defined in the section tag. Records created in SEER*Abs cannot have sub-entities; therefore, the repeating attribute is not valid in data entry layouts.

section

The section tag encapsulates a set of rows. You must define at least one section on each tab. Multiple sections may be defined at the same level; and sections may be nested within sections. To nest a section you must define the inner section within a row (please refer to the XML Structure for Layouts provided at the beginning of this chapter). You can use nesting to define sub-entities as described in the description of the repeating attribute. You can also use nesting to display a set of rows in a title bordered using different indentation. The section tag has the following attributes:

- label (optional): If a label is defined, the section is shown with a frame border and the label is used as the section title. No border is displayed when a label is not defined.

- repeating (optional, only applicable in patient set layout): This identifies a sub-entity that can have multiple instances within an entity defined at a higher level. The higher level may be an outer section, this section's tab, or the main layout. The repeating attribute defines data structure and display. If this section is nested within another section that has a repeating attribute then this sub-entity is also nested in terms of data structure. The path in the import and load scripts must include the full path defined in the layout. Records in SEER*Abs cannot have sub-entities; therefore, this attribute is not valid in record layouts.

- indentation (optional): The number of pixels of white space used to indent the section. If not specified, a default value of "0" will be used.

row

The row tag encapsulates a set of fields and/or a section. Sections nested within sections must be embedded within a row as described above.

The row tag has the following attributes:

- gap-before-row, gap-after-row (optional): the gap (in pixel) above and below this row. If not specified, the default values set for this layout are used (see *default-gap-before-row* and *default-gap-after-row* attributes of the editor-layout tag).

field

The field tag defines the properties within this layout's entity. The attributes within the field tag define the data item within the database and define its display properties. Tags embedded within the field tag define logic associated with the data item (on-entity-opened, on-entity-saved, and on-field-exited).

- name (required, cannot be blank): This is the database name for the field; this value must be unique within the layout. This field name is used in all references to this data item. For data entry layouts, the field name is used as a reference in extract scripts, on-entity scripts, and on-field scripts. For reference data entities, this field name is used to reference the data item in search layouts, filters, load scripts, and import scripts. An entity-to-field mapping is used in scripts. The format of the mapping in Groovy scripts is entity['fieldname']. To view an example, review the layout and script shipped with SEER*Abs for NAACCR reference records. The layout includes a field with name = "nameLast". The Groovy mapping syntax for this field in the load script is "record['nameLast']".

- label (required, cannot be blank): The field label that is displayed on the screen. This attribute cannot be blank because it is used by the edits to reference a field on the screen. To display a field without a label, set the *show-label* attribute to false.

- case (optional): If set to "upper" or "lower", text entered for the field will be auto-converted to the specified case. If set to "as-is", the case of the text will be as entered. If case is not specified for a field, the *default-case* assigned in the editor-layout tag is used.

- editable (optional): If false, the field will be displayed in read-only mode. If not specified or set to true, the user will be able to enter values for the field.

- gap-before-label (optional): The number of pixels of white space displayed before the field's label. If not specified, the *default-gap-before-label* assigned in the editor-layout tag is used.

- gap-after-label (optional): The number of pixels of white space displayed after the field's label. If not specified, the *default-gap-after-label* assigned in the editor-layout tag is used.

- length (optional): This is the maximum number of characters that can be entered for the field. Length is not used for unlimited-string fields. If length is not specified, a default value of "10" is used for date fields, a default value of "1" is used for other types of fields. The length does not determine the width of the text box displayed on the screen, the size of the box is determined by the *shown-col* attribute.

- lookup (optional): If a lookup is available for the field, the lookup ID is specified in this attribute (lookup IDs are listed in the Lookup Definitions configuration file). A light bulb is displayed next to the field. Click the light bulb or use the CTRL-L shortcut to open the lookup. This attribute is ignored if the data type is unlimited-string.

- searchable (optional): This attribute must be set to "true" in order to use this as a search field on the Search page. When a field is searchable, the field is indexed in the database. If a non-searchable field is made searchable, you must recreate the indexes in all databases to use the new search field (File > Recreate Indexes). Default value is 'false' if none is provided.

- searchable-as (optional): This is the name used to index this field. If this is not specified, the field name is used. This attribute allows you to specify a different search name so that you can use one search field to search multiple fields (for example this is set to "race" for 'race1', 'race2', etc.).

- shown-col (optional): The field's text box will be wide enough to show this number of characters. The number of characters that can actually be entered into the box equals the length attribute. If the number entered is greater than shown-col, the arrow keys can be used to scroll through the text. If shown-col is set to 0, no box will be shown on the page (use 0 if you only need to display a label on the screen). If no value is provided for shown-col, the field's default width will be used: the width of an unlimited-string field is only limited by the contents of its row; the width of all other data types equals the length attribute.

- shown-row (optional): This is the number of rows used to display an unlimited-string field. If no value is provided for shown-row in a data entry layout, a single row is displayed. If no value is provided for shown-row and the layout is read-only, the box height will auto-adjust to the number of rows required to display the current text.

- trim (optional): This is the rule used to remove leading and trailing white space from the field. The available values are 'right', 'left', 'both' and 'none'. If not specified, the value of *default-trim* in the editor-layout tag is used.

- autocomplete-list (optional): This identifies the list of terms that will be presented if the abstractor uses autocomplete to enter data in this field. If not specified, the list named "default" will be used.
- type (optional): This is the field's data type. If not specified, string is used by default. The following types may be used:
    - string: the field can have any value and is displayed in a single-line text box
    - number: the field can only contain digits
    - boolean: the field con only contain "1" or "0"
    - date: the field must have a value in the format of mm/dd/yyyy. The length attribute does not need to be specified, it defaults to 10 for date fields.
    - facility: must contain a valid facility display ID ('FAC-XXXX'); use the lookup "*lkup_internal_facility*" to bind the field to the special facility lookup.
    - physician: must contain a valid physician display ID ('PER-XXXX'); use the lookup "*lkup_internal_physician*" to bind the field to the special physician lookup.
    - unlimited-string: the field can have any value and it will be shown as a multi-row text box with the label above the box. The width and number of rows are defined by the shown-row and shown-row attributes, respectively. The field is defined as a CLOB in the database, the length attribute is ignored.
    - checkbox: the field will be displayed as a checkbox that the user can check or uncheck; correspond to the Boolean values of "1" and "0". Mainly used in the user-input layouts.
    - dropdown: the field will be displayed as a drop-down box where the user can select a particular label but can still type any free text. Dropdown fields must be bind to a lookup; only the labels of that lookup will be shown in the choices while the corresponding code will be saved in the entity when the user selects a label (labels must be unique for that purpose). Dropdown fields are mainly used in search criteria in the search module.

on-entity-opened

This tag is optional and contains a Groovy script that is run when the entity is opened in the editor. The script should not return any value (if it does, it will be ignored).

on-entity-saved

This tag is optional and contains a Groovy script that is run when the entity is saved in the editor. The script should not return any value (if it does, it will be ignored).

on-field-exited

This tag is optional and contains a Groovy script that is run when a field loses focus. The script should not return any value (if it does, it will be ignored).

desc

This tag is optional and provides documentation about the field. That documentation will be shown to the user when he/she clicks the field's label or type Ctrl+N. The description should be provided as simple HTML text (no tables, images or CSS). It is recommended to use "<![CDATA[" and "]]>" to avoid conflicts with the XML tags. See the abstract layout for example of descriptions.

Section 7: Defining User-input Layouts

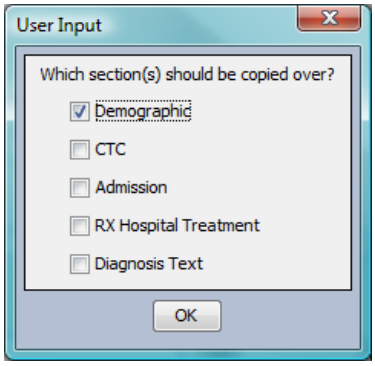
User-input layouts are very similar to regular layouts; the difference is that they can be added, removed or modified without any consequences in the application; while only the content of a regular layout can be modified (for example the layout for AFL is the *layout-afl.xml* file located in the configuration directory; deleting that file outside of the application will result in a failure during the startup process). Because the user-input layouts can be added and removed, they have their own directory (conf/layouts/).

The XML syntax for the user-input layout is exactly the same as the regular layouts and is not repeated in this section.

Once a user-input layout has been defined, it can be referenced by any non-embedded script using its file name as a parameter to a utility method:

```
input = env.utils.showInputDialog('input-abstract-to-abstract.xml', false)
```

The second parameter to that utility method is a Boolean that should be set to true if the cancel button should appear in the input dialog. When the script is executed and reaches that line, a dialog will be presented to the user and the script execution will block. The user will have the opportunity to fill-in some values on the dialog and click OK or Cancel (if the cancel button has been enabled). Here is an example of a user-input dialog that is used when copying an abstract into a new abstract; it requests the user which fields should be copied over (note that this particular layout uses only check-boxes but regular free-text can also be used):



The result of the call is a map containing the user input (the keys are the field names defined in the layout and the values is what the user typed (the absence of a key in the map should be considered as a blank value). That map can then be used in the script to take decisions (in this case which fields to copy over).

To add a user-input layout:

1. Log in as the administrator.
2. Open the Configuration module.
3. Click the Add Layout button in the Input Dialog Layouts section.

4. Provide a file name for the new layout; **do not include the file extension (.xml)**; by default SEER*Abs will use the prefix *input-* for the file name but that is not a requirement and can be changed. If a description is provided it will be shown in the Input Dialog Layouts table.

5. Click the OK button, the new layout will appear in the Input Dialog Layouts table. When creating a new user-input layout, SEER*Abs uses a default empty layout as a first template. Double click the corresponding row to edit it.

6. Customize the layout. Use the preview tab to see the result of your customization.

7. Close the editor; if prompted to save your changes, click Yes. The new layout will automatically be reloaded and made available to the rest of the application. It is now ready to be referenced in any non-embedded script.

To delete a user-input layout:

1. Log in as the administrator.

2. Open the Configuration module.

3. Locate the row containing the layout in the Input Dialog Layouts table.

4. Click the delete icon of that row.

5. When prompted for confirmation, click the Yes button.

Section 8: Configuring Searches & Filters

XML configuration files are used to define screen layouts for the Worklist table, User account manager, the three tabs of the Search page, the facility lookup, and the physician lookup. These configuration files are defined as layouts with two sections: criteria-layout defines the filters and the table-layout defines the table in which the results are displayed. Filters cannot always be defined. The search and filter layout files are listed below.

- **Facility** (search-facility.xml) – the layout of the Facility tab of the Search page.
- **Facility Lookup** (search-facility-lkup.xml) – the layout of the internal lookup for facilities. A single search box is shown in that lookup; for that reason the criteria defined in the configuration file is not used to show different search fields, but instead it is used to know which fields should be searched when the user types text in the unique search box.
- **Patient Data** (search-patient.xml) – the layout of the Patient Data tab of the Search page.
- **Physician** (search-physician.xml) – the layout of the Physician tab of the Search page.
- **Physician Lookup** (search-physician-lkup.xml) – the layout of the internal lookup for physicians. A single search box is shown in that lookup; for that reason the criteria defined in the configuration file is not used to show different search fields, but instead it is used to know which fields should be searched when the user types text in the unique search box.
- **Users** (search-user.xml) – the layout of the Users Account manager. No criteria can be defined for that configuration file.
- **Worklist** (search-worklist.xml) – the layout of the Worklist. The filter in the worklist cannot be customized but it does use a free-text search box. The criteria defined in this configuration file is used to know which fields should be searched when the user types text in that free-text search box.

The searches in SEER*Abs are implemented using an external library called Lucene (http://lucene.apache.org/java/docs).

To modify a search layout:

1. Log in as the administrator.
2. Open the Configuration module.
3. Locate the layout that needs to be modified in the Searches and Filters section.
4. Double-click the row containing that layout to open it in the Configuration File Editor.
5. Modify the XML; use the preview tab to see the changed criteria (some layouts do not use the criteria on the screen and the preview is not relevant); the results table is also shown with two empty rows.
6. Once done, close the editor.

7. If prompted to save the changes, click Yes.

8. The search layout is automatically reloaded in the application.

XML Structure for Searches & Filters

```
<search-layout>
    <criteria-layout>
        <row>
            <field/>
        </row>
    </criteria-layout>
    <table-layout>
        <column>
            <on-field-populated/>
        </column>
    </table-layout>
</search-layout>
```

search-layout

This is the root XML tag. It is required and can only be defined once. It has the following attributes:

- max-num-results: the maximum number of results that the search can return (use -1 for no maximum).
- desc: a short description for the configuration file.

criteria-layout

Use the criteria-layout tag to define the filters shown at the top of the physician lookup, facility lookup, and the search tabs. This tag is ignored in the layouts for the Worklist and User Account manager. The criteria-layout tag has the following attributes:

- default-case (optional): If this attribute's value is upper or lower, the text entered in each search field will be converted unless a case attribute is specified in the field tag. The search algorithms are not case sensitive. Valid values for this attribute are 'upper', 'lower' and 'as-is'. If a value is not specified, "upper" will be used by default.
- default-trim (optional): Leading and/or trailing white space will be removed as specified by this rule. The following values are valid: "right", "left", "both" and "none". If a value is not specified, "right" is used as the *default-trim*. You may over-ride this attribute by setting the *trim* attribute for individual fields

- default-gap-before-label (optional): The number of pixels of white space displayed to the left of each search field label. 10 pixels are used if this attribute is not specified. You may over-ride this attribute by setting the *gap-before-label* attribute for individual fields.
- default-gap-after-label (optional): The number of pixels of white space displayed to the right of each search field label. 3 pixels are used if this attribute is not specified. You may over-ride this attribute by setting the *gap-after-label* attribute for individual fields.
- default-gap-before-row, default-gap-after-row (optional): The number of pixels of white space displayed above and under each row of search fields. If not specified, 5 pixels are used by default for each attribute. You may over-ride these attributes by setting the *gap-before-row* and *gap-after-row* attributes for individual rows. The total space between row1 and row2 = (gap after row1 + gap before row2).

row

Multiple rows of search fields can be displayed. The row tag contains a set of fields and has the following attributes:

- gap-before-row, gap-after-row (optional): the gap (in pixel) before and after this row. If not specified, the default values set for this layout are used (see *default-gap-before-row* and *default-gap-after-row* attributes of the criteria-layout tag).

field

The field tag defines the properties used to search. The field tag has the following attributes:

- name (required): This is the database name for the field. In order for a search to be successful for this field, this value must correspond to the *name* or *searchable-as* attribute of a *searchable* field. Searchable fields are defined in the layout or layouts associated with the search (for example, reference record and patient data layouts are all associated with the Patient Data search layout).
- label: The field label that is displayed on the screen. If not label is provided, the default “Label” is used.
- show-label: if false then the label won’t be shown on the screen.
- length (optional): This is the maximum number of characters that can be entered for the field. If length is not specified, a default value of “10” is used for date fields; a default value of “1” is used for other types of fields. The length does not determine the width of the text box displayed on the screen, the size of the box is determined by the *shown-col* attribute.
- type (optional): the type of the field value. The available types are
    - string: the field can have any value and is displayed in a single-line text box
    - number: the field can only contain digits
    - boolean: the field con only contain “1” or “0”

- date: the field must have a value in the format of mm/dd/yyyy. The length attribute does not need to be specified, it defaults to 10 for date fields.
- facility: must contain a valid facility display ID ('FAC-XXXX'); use the lookup "*lkup_internal_facility*" to bind the field to the special facility lookup.
- physician: must contain a valid physician display ID ('PER-XXXX'); use the lookup "*lkup_internal_physician*" to bind the field to the special physician lookup.
- checkbox: the field will be displayed as a checkbox that the user can check or uncheck; correspond to the Boolean values of "1" and "0".
- dropdown: the field will be displayed as a drop-down box where the user can select a particular label but can still type any free text. Dropdown fields must be bind to a lookup; only the labels of that lookup will be shown in the choices while the corresponding code will be saved in the entity when the user selects a label (labels must be unique for that purpose).

- shown-col (optional): The field's text box will be wide enough to show this number of characters. The number of characters that can actually be entered into the box equals the length attribute. If the number entered is greater than shown-col, the arrow keys can be used to scroll through the text. If shown-col is set to 0, no box will be shown on the page (use 0 if you only need to display a label on the screen). If no value is provided for shown-col, the field's default width will be used: the width of an unlimited-string field is only limited by the contents of its row; the width of all other data types equals the length attribute.
- editable (optional): If false, the field will be displayed in read-only mode. If not specified or set to true, the user will be able to enter values for the field.
- case (optional): If set to "upper" or "lower", text entered for the field will be auto-converted to the specified case. If set to "as-is", the case of the text will be as entered. If case is not specified for a field, the *default-case* assigned in the criteria-layout tag is used.
- trim (optional): This is the rule used to remove leading and trailing white space from the field. The available values are 'right', 'left', 'both' and 'none'. If not specified, the value of *default-trim* in the criteria -layout tag is used.
- gap-before-label (optional): The number of pixels of white space displayed before the field's label. If not specified, the *default-gap-before-label* assigned in the criteria -layout tag is used.
- gap-after-label (optional): The number of pixels of white space displayed after the field's label. If not specified, the *default-gap-after-label* assigned in the criteria -layout tag is used.
- lookup (optional): If a lookup is available for the field, the lookup ID is specified in this attribute (lookup IDs are listed in the Lookup Definitions configuration file). A light bulb is displayed next to the field. Click the light bulb or use the CTRL-L shortcut to open the lookup.

- search-type (required): the type of search that should be used for this field, available values are “exact”, “contains” and “fuzzy”. Fuzzy match results include partial and pattern-based matches. The fuzzy search algorithms are implemented via the Apache Lucene text search library (see the documentation section of http://lucene.apache.org/java/docs for more information). If fuzzy matching is used, the results are sorted to show matches with the highest scores first.

table-layout

This tag is used to configure the table containing the results of a search. It contains a collection of columns and has no attributes.

column

This tag represents a single column in the search results table. The values in the column can be formatted via a script defined in an on-field-populated tag. The column tag has the following attributes:

- title (required) - the column header.
- name (required) - the database name for the field displayed in the column. It must correspond to the *name* attribute of a field in the corresponding layout (or layouts).
- center (optional) - If set to “true”, the values displayed in the column will be centered. If “false”, the values will be left-justified. If not specified, “false” is used by default.
- width (optional) - the initial size of the column. Possible values are: “min” - use as little space as possible, based on the data in the column; “max” - used as much space as possible; and “fixed” - column should not be resizable and will always have the same width; see 'length' attribute). Default is 'max' if none is provided.
- length (optional) - The actual size (width) of the column in pixels. This attribute is only used if the width attribute is set to “fixed”.
- lookup (optional) - If a lookup is available for the field, the lookup ID is specified in this attribute (lookup IDs are listed in the Lookup Definitions configuration file). The column data are formatted based on this lookup.
- default (optional) – default to use when a value for this particular column is blank.
- default-sort (optional) – by default, use this column as a default sorting on the table; available values are “ascending” and “descending”.

on-field-populated

This tag is optional and contains a Groovy script that is run for all values in the column. The script must return a string value (or null). The value returned by the script is the formatted value displayed in the column. Keep in mind that running a script on each value of a particular column for a large table can be slow and resource demanding.

Section 9: Defining Scripts

The Scripts section of the Configuration Manager contains Groovy scripts to create extract files, load reference data from external files or directly from a database, and utility scripts used by the system to support AFLs and edits. Groovy is a scripting language for the Java platform. The Internet has several Groovy references including the Groovy home page at http://groovy.codehaus.org. The official site contains a lot of information, including tutorials for people new to Groovy.

SEER*Abs scripts are listed in the Configuration Manager; because so many scripts are used, they have been grouped by family. The Family column in the Scripts table contains the following values:

- **Action**: scripts that are run as a result of a user action (creating a record, saving a record, updating an AFL, etc…)
- **Extract:** scripts that create export files containing AFL updates or records created in SEER*Abs. The extract files are used to update the registry's data management system with the new data.
- **Import:** scripts that load reference data from files.
- **Load:** scripts that load reference data via direct queries to the registry's main database.
- **Upload:** scripts that transfer data directly from SEER*Abs to the registry's main database. The default configuration provided with SEER*Abs does not contain an implementation for those scripts.
- **Generic:** scripts that are not part of any other families.

The data types are AFL, physician, facility, patient (indicating that it the script deals with patient sets), and record. Records also list a subtype in parenthesis. The Generic family usually doesn't apply to a particular type.

To modify a script:

1. Log in as the administrator.
2. Open the Configuration module.
3. Locate the script that needs to be modified in the Scripts section.
4. Double-click the row containing that script to open it in the Configuration File Editor.
5. Modify the Groovy code; use the Validate button to make sure the code is valid. There is currently no way to validate the logic of the code; the script has to be tested in the application for that purpose.
6. Once done, close the editor.
7. If prompted to save the changes, click Yes.

8. The script is automatically reloaded in the application.

Section 10: Defining Action Scripts

Action scripts are very similar to regular scripts; the difference is that they can be added, removed or modified without any consequences in the application; while only the content of a regular script can be modified (for example the extract script for abstract is called *script-extract-record-abstract.groovy*; deleting that file outside of the application will result in a failure during the startup process). Because the action scripts can be added and removed, they have their own directory (conf/scripts/).

Writing action script is not different than writing regular scripts; the scripting language (Groovy, see http://groovy.codehaus.org) is identical. One minor distinction is that the regular scripts usually receive data in their context (for example, the script that runs when a record is saved receives that record in its context so it can be modified by the script) while the action script do not receive any data in their context (since there are triggered by the user selecting them from a menu item).

Once an action script has been defined, it is available in the *Action* menu. The default configuration provided with SEER*Abs contains a single action script called “Purge Entities”; it deletes from the database any AFL or RECORD that have a status of ARCHIVED.

To add an action script:

1. Log in as the administrator.
2. Open the Configuration module.
3. Click the Add Script button in the Action Scripts section.
4. Provide a file name for the new layout; **do not include the file extension (.groovy)**; by default SEER*Abs will use the prefix *action-* for the file name but that is not a requirement and can be changed. A menu label must be provided; this is the label as it will appear in the menu item under the *Action* menu. If a description is provided it will be shown in the Action Scripts table.
5. Click the OK button, the new script will appear in the Action Scripts table. When creating a new action script, SEER*Abs uses a default empty script as a first template. Double click the corresponding row to edit it.
6. Customize the script.
7. Close the editor; if prompted to save your changes, click Yes. The new script will automatically be reloaded and made available to the rest of the application. It is now ready to be called from the *Action* menu.

To delete an action script:

1. Log in as the administrator.
2. Open the Configuration module.

3. Locate the row containing the script in the Action Scripts table.
4. Click the delete icon of that row.
5. When prompted for confirmation, click the Yes button.

Section 11: Defining Lookups

Lookup tables provide a list of valid values for a field. Typically, this is a list of codes and a user-friendly description of the code. When a lookup is associated with a field in a layout, a light bulb is displayed next to the field. The lookup table is displayed when the user clicks the light bulb, they may then select a value from the list.

SEER*Abs lookup tables are defined in a single configuration file (lookups.xml). This file includes definitions for lookups that you create and internal lookups required by the system ("lkup_internal" prefix). All internal lookups are required. You may add or modify the entries in a few of the internal lookups. Please refer to the comments in the Lookup configuration file for more information related to the internal lookups.

To modify, add or delete a lookup:

1. Log in as the administrator.
2. Open the Configuration module.
3. Double-click the unique row of the Lookups section to open the corresponding file in the Configuration File Editor.
4. Modify the file; make sure you take into account the comments on the top of the file as some lookups cannot be modified while others can be modified but not removed.
5. Once done, close the editor.
6. If prompted to save the changes, click Yes.
7. The lookups are automatically reloaded in the application.

XML Structure for Lookups

```
<lookup-definitions>
    <lookup>
        <entry/>
    </lookup>
</lookup-definitions>
```

lookup-definitions

This is the root XML tag for Lookup Definitions. This tag is required and can only be defined once in the file. There are no attributes for the lookup-definitions tag.

lookup

The lookup tag defines a single lookup in the application. It contains a collection of entries. The lookup tag has the following attributes:

- id (required) – This is the lookup's name, it must be a unique identifier. A lookup is assigned to a field by specifying this ID in the field's lookup attribute.
- external (optional): whether or not this lookup is defined in the XML file or is loaded through one of the scripts from the synchronization module. The available values are 'true' and 'false'. Default value is 'false' if none is provided.
- disable-strict-validation (optional): whether or not strict validation should be disabled for this lookup. If strict validation on lookups is on, a wrong value typed in a field that contains a lookup will generate a critical error. Strict validation on lookups can be turn on and off in the main configuration. If strict validation is off, this attribute won't have any effect. If it is on and this attribute is set to "true" for a particular lookup, then strict validation will be disabled for that lookup. It is useful to disable the strict validation for some lookups that have combined values. For example, an histology lookups could have codes that are a combination of the histology and behavior ("8000/2"); when selecting a value, the "8000" part should be assigned to the histology field while the "2" part should be assigned to the behavior field. Such a lookup would be bind to the histology field, but strict validation would always fail because the histology value by itself would never be found in the lookup. The available values for this attribute are 'true' and 'false'. Default value is 'false' if none is provided.

entry

The entry tag defines the mapping between the true value (the code) and the formatted text (the label). As an example, consider a mapping in a race lookup: code="01", label="race". The entry tag has the following attributes:

- code (required) – absolute value that will be entered in a field
- label (required) – formatted text that describes the code

Section 12: Defining Edits

Computerized edits are integrated into SEER*Abs to test the validity of data. In SEER*Abs, the edits are executed on records created in SEER*Abs, they are not executed on reference data. The following sets of edits are available in SEER*Abs:

- Internal system edits enforce data type constraints in layouts. The system edits cannot be modified.
- SEER*Abs edits are defined and maintained by registry staff. Samples are provided in the configuration file shipped with SEER*Abs. SEER*Abs edits apply to any record types created in the application.
- SEER Edits cover fields submitted to SEER and represent the edits implemented in the SEER*Edits software. The SEER edits are defined in XML files provided and maintained by the SEER*Edits development team. The SEER edits configuration file cannot be modified. By default SEER*Edits are not loaded in SEER*Abs but that behavior can be changed through a configuration variable in the main configuration file. If loaded, the SEER*Edits are applied only to abstract records.

SEER*Abs edits are implemented in Groovy, the scripting language for the Java platform that is also used for SEER*Abs scripts. Edits uses a small subset of the Groovy syntax. A working knowledge of regular expressions and Groovy logic statements are needed to maintain edits in SEER*Abs. To define a new edit, it is recommended that you copy-and-paste the code from an existing edit and use that code as a template.

Guidelines for writing the Groovy code for a registry edit:

- An edit error is triggered if the code returns FALSE for the record or patient set. The edit passes if the code returns TRUE.
- Use the Groovy code of a similar edit as a template.
- Contexts are defined within the context tag of the XML. Many examples are provided in the SEER edits XML file. Your Groovy code may include references to contexts that you define and the contexts defined for the SEER edits.
- A context is a Java naming system. Contexts are used to define arrays, hash tables, and functions used by the edits. For example, there are a large number of contexts defined for the SEER*Edits. Primarily, these represent data tables required by the SEER Edits logic. Contexts are defined within the XML context tag. Many examples are provided in the SEER edits XML file. Your Groovy code may include references to contexts that you define and the contexts defined for the SEER edits.

Section 13: Defining Autocompletion Word Lists

While entering text in a field, an abstractor may press Ctrl+Space to use the autocomplete feature. Autcomplete is available for string and unlimited-string fields in record and AFL layouts.

SEER*Abs supports multiple sets of autocomplete terms. Separate lists may be designated for different fields, for example, there may be one list for histology and another for primary site. Or terms from all lists may be made available in a field, for example, all terms are typically made available when editing large text fields. When a field does not define any autocomplete list in the layout definition, it will automatically use the "default" list. For that reason, there must always be a list with a name "default" define in the autocompletion word lists.

To modify, add or delete a autocompletion lists:

1. Log in as the administrator.
2. Open the Configuration module.
3. Double-click the unique row of the Autocomplete Terms section to open the corresponding file in the Configuration File Editor.
4. Modify the file; use the Validate button to make sure the XML is correct.
5. Once done, close the editor.
6. If prompted to save the changes, click Yes.
7. The lists are automatically reloaded in the application.

XML Structure for Autocompletion Word Lists

The following the available XML tags and their hierarchy in the Autocomplete configuration file.

```
<auto-complete-terms>
    <list>
       <import-list/>
       <term/>
    </list>
</ auto-complete-terms >
```

auto-complete-terms

This is the root XML tag. It is required, can only be defined once, and has no attributes.

list

Multiple lists may be defined. Each list may contain import-list tags and/or term tags; or you may define an empty list. The list tag has one attribute:

- name (required): The list identifier. Layout scripts may associate a specific list with a field using the autocomplete-list attribute of the field tag. You must define a list named “default” which will be used for fields that do not have an autocomplete-list attribute (the default list may be empty, but it must exist).

import-list

Use this tag to create one list based on other lists. In the example below, the default list does not have its own set of terms but it inherits the terms from the histology and site lists.

```
<list name=”default”>
   <import-list>histology</import-list>
   <import-list>site</import-list>
</list>
```

term

Use this tag to define a term. Single words or phrases may be used.

Section 14: Managing User Accounts

SEER*Abs supports a single administrative user account (username = admin) and multiple abstractor accounts. The admin user account is created during the initial installation on the administrator's computer. The system administrator configures SEER*Abs and creates a registry-specific installation file. Registry managers and the SEER*Abs system administrator must define a protocol for maintaining abstractor accounts.

- A single abstractor account may be created as SEER*Abs is installed on each workstation. The abstractor using that computer would then complete the installation by defining a password known only to them.
- Alternatively, the system administrator may create accounts for all abstractors during the initial configuration. A protected list of unique passwords would be created. Accounts for all abstractors would then be installed on all workstations.

There is no method for synchronizing the user accounts on multiple installations of SEER*Abs. Once the system is deployed, you will need to add and remove users from each installation; or modify a central version and re-install the software.

To add, modify, or delete user accounts:

1. Login as the administrator.
2. Open the Users Manager.
3. To add a new account, click **Add User**.
    a. Provide a username and the initial password for the new user. The password must contain at least 1 lower-case letter, 1 upper-case letter, and either a digit or a special character. This is the password that the user will enter the first time they login. They will then be prompted to specify a password known only to them.
    b. You may enter values for the user-defined fields (optional). These fields are defined in the User layout configuration files.
    c. Click **Save**.
4. To delete a user account, click the X icon associated with the user account. The admin account cannot be deleted.
5. To define a new password for an account:
    a. Double-click the username or click the edit icon.
    b. Click **Change Pswd**
    c. Enter the new **Password**.
    d. Verify the new password by re-entering it into the **Repeat Password** field.

6. To set or change the values of registry fields:

    a. Double-click the username or click the edit icon.

    b. Enter new values for the registry defined fields.

Section 15: SEER*Abs and Sensitive Data

SEER*Abs is a tool designed to handle sensitive data. The confidentiality of that data is critical. A lot of resources can be found online about good practices when it comes to handle sensitive data; one of those resources is the NIH website (http://www.nih.gov/).

SEER*Abs provides the following mechanism to keep the data secured:

- Strong passwords: any passwords defined for the admin user or regular users must be at least 8 characters long and must contain 1 lower-case letter, 1 upper-case letter and either a digit or a special character.
- Database encryptions: Derby is a file-based Java database; the content of the tables is saved as binary data and cannot be read through any text editor. Those binary files are also encrypted by Derby using the Data Encryption Standard (DES) 56 bits algorithm. That algorithm is tied to a key that is required to make any interaction with the databases. Without the key, any external programs (implemented in Java or other languages) will be unable to create a connection to the database.

When evaluating the data confidentiality aspects in SEER*Abs, keep the following in mind:

- SEER*Abs does not enforce changing the password after a certain period of time; it should be a Registry policy to do so.
- Once an extract has been generated by SEER*Abs, it will be located in the output directory (or another directory if the user changed that default one). It won't be encrypted and readable by any text editor. It is the user's responsibility to move the file right away to a secured location in the Central Registry. Note that the extraction is done by script, which means that even if the default scripts provided with SEER*Abs do not encrypt the content of the files, they can be modified by the Registry to do so.
- When synchronizing by directly accessing another database, SEER*Abs creates a connection to that database. Any data going through that connection is NOT encrypted. That means the synchronization needs to be done at the Central Registry (behind a firewall), or through a secured VPN.
- The database encryption is not unbreakable (and to some extend is not very strong). It is recommended to encrypt the entire hard-drive of the laptop (or any other device used to run the application).

Appendix 1: Utility Functions for Scripts

convertDate

This method can be used in any script. Converts the incoming date into a string representation of the corresponding java time.

param1: date to convert, format must be 'yyyymmddhhmmss'

deleteEntities

This method can be used in any script. Deletes all the entities for the passed type and subtype (admin user cannot be deleted through this method)

- param1: Entity type (required)
- param2: Entity subtype (use null if not applying)

deleteEntities

This method can be used in any script. Deletes the passed entity (admin user cannot be deleted through this method)

- param1: Entity type (required)
- param2: Entity subtype (use null if not applying)
- param3: Entity to delete

deleteEntity

This method can be used in any script. Deletes the passed entity (admin user cannot be deleted through this method)

- param1: Entity type (required)
- param2: Entity subtype (use null if not applying)
- param3: Entity to delete

disableButton

This method can be used only in scripts from the 'Action' family (on-record-created, etc...). Disables the passed button in the current editor (button will be re-enabled once the editor is closed)

- param1: requested button

displayEnv

This method can be used in any script. Displays the available environment variables in the current progress window or in the log if no window is available.

formatDateValue

This method can be used in any script. Formats the passed date value (a string representation of the Java time, see 'System.currentTimeMillis()'); result is mm/dd/yyyy.

- param1: value to format
- param2: if true, the hours and minutes will be included

formatValue

This method can be used in any script. Formats the passed value using a lookup; if the lookup does not contain the value it is returned as-is.

- param1: value to format
- param2: existing lookup ID

getConfVariable

This method can be used in any script. Returns the value of the requested configuration key.

- param1: key from the main configuration file (seerabs.properties)

getCurrentDay

This method can be used in any script. Returns the current day as a string; value is left-zero padded to be 2 characters long.

getCurrentFacility

This method can be used in any script. Returns the current facility as a map of property/values or null if no facility was used during the login process.

getCurrentMonth

This method can be used in any script. Returns the current month as a string; value is left-zero padded to be 2 characters long.

getCurrentUser

This method can be used in any script. Returns the currently logged user as a map of property/values.

getCurrentValue

This method can be used only in scripts embedded in XML layouts. Returns the value (possibly null) of the field to which this Groovy snippet is attached

getCurrentValue

This method can be used only in scripts embedded in table definition files. Returns the value (possibly null) of the column to which this Groovy snippet is attached for the entity displayed in the current row.

getCurrentYear

This method can be used in any script. Returns the current year as a string.

getEntityByDisplayId

This method can be used in any script. Returns the entity corresponding to the passed display ID or null if it is not found.

- param1: Entity type (required)
- param2: Entity subtype (use null if not applying)
- param3: display ID

getLookupById

This method can be used in any script. Returns the lookup (map of code/value) corresponding to the requested ID, throws exception is lookup is not found or empty.

- param1: Lookup ID

getPropertiesFromLayout

This method can be used in any script. Returns the properties defined in the layout for the pased type and subtype

- param1: Entity type (required)
- param2: Entity subtype (use null if not applying)
- param3: if true the read-only properties will be included, they won't otherwise

getRawLookupValue

This method can be used only in scripts embedded in XML layouts. Returns the raw value that was selected by the user; useful in cases where the codes define in the lookup are a combination of several values (for example histology/behavior represented as 8000/2)

getSeerabsVersion

This method can be used in any script. Returns the application current version number.

getValue

This method can be used only in scripts embedded in XML layouts. Returns the value (possibly null) of the requested field

- param1: requested field name

getValue

This method can be used only in scripts embedded in table definition files. Returns the value (possibly null) of the requested field for the entity displayed in the current row.

- param1: requested field name

hideButton

This method can be used only in scripts from the 'Action' family (on-record-created, etc...). Hides the passed button in the current editor (button will be visible again once the editor is closed)

- param1: requested button

jumpToField

This method can be used only in scripts embedded in XML layouts. Changes the focus to be on the requested field

- param1: requested field name

leftPad

This method can be used in any script. Left-pads the passed value.

- param1: the value to pad
- param2: the number of characters the result needs to be
- param3: the string to use as a padding character

log

This method can be used in any script. Display the passed message in the current progress window (if there is one) and in the log.

- param1: line to display

rightPad

This method can be used in any script. Right-pads the passed value.

- param1: the value to pad
- param2: the number of characters the result needs to be
- param3: the string to use as a padding character

saveEntities

This method can be used in any script. Saves the passed entities (admin user cannot be saved through this method)

- param1: Entity type (required)
- param2: Entity subtype (use null if not applying)

- param3: List of entities to save

saveEntity

This method can be used in any script. Saves the passed entity (admin user cannot be saved through this method)

- param1: Entity type (required)
- param2: Entity subtype (use null if not applying)
- param3: Entity to save

searchEntities

This method can be used in any script. Returns the entities corresponding to the passed search criteria.

- param1: Entity type (required)
- param2: Entity subtype (use null if not applying)
- param3: search criteria, see uses Lucene syntax (http://lucene.apache.org/java/docs).
- param4: max result (-1 for all results)

setCurrentValue

This method can be used only in scripts embedded in XML layouts. Sets the value of the field to which this Groovy snippet is attached

- param1: value to set

setEditorToReadOnly

This method can be used only in scripts from the 'Action' family (on-record-created, etc...). Sets the current editor in read-only mode (that mode will be reset once the editor is closed)

setValue

This method can be used only in scripts embedded in XML layouts. Sets the value of the requested field

- param1: requested field name
- param2: value to set

showConfirmationDialog

This method can be used in any script. Shows a confirmation dialog with a customized message and a yes/no option; returns true if the user accepted, false otherwise.

- param1: message to display

showInputDialog

This method can be used in any script. Shows an input dialog based on a customized input layout filename; returns the input values as a map of field name/field value; one of those value is 'closingStatus' which is set to 'ok' or 'cancel' depending which button the user clicked. If a field is not returned, null should be assumed for its value.

- param1: input layout filename
- param2: true if the cancel button should be shown, false otherwise

updateEntities

This method can be used in any script. Updates the passed entities (admin user cannot be updated through this method)

- param1: Entity type (required)
- param2: Entity subtype (use null if not applying)
- param3: Entity to update

updateEntity

This method can be used in any script. Updates the passed entity (admin user cannot be updated through this method)

- param1: Entity type (required)
- param2: Entity subtype (use null if not applying)
- param3: Entity to update

updateLookup

This method can be used in any script. Saves or updates the passed lookup

- param1: lookup ID
- param2: lookup content (map of code/label)