Fundamentals of Data Synchronization

Installations of DDMS can share data with each other using the data synchronization feature. No connection to the Internet or other network is required. Data is exchanged by the sharing of data files between installations. These data files can be distributed on a compact disk, a flash drive, or any other kind of data media. Each installation can produce a data file that contains all records that were created or updated on that installation. Any other installation that imports this data file will then contain these same records.

Each installation of DDMS tracks all of its transactions, which can then be sent to other installations using the synchronization feature. Examples of tracked transactions include data entry through the web interface, excel imports, and synchronization data imported from other installations. Each transaction record is a self-contained unit, containing all information required to duplicate itself on other computers. Since records are created locally when importing transaction data from other installation, multiple updates can be easily combined into a single export file. For example, if computer A exports data to computer B, and computer B then exports its data to computer C, C will get the data of both A and B.

For example: A village exists within a district, which exists within a province, which exists within a country. The village installation exports its data to the district installation. The district installation now contains its data plus the data from the village. The district then exports its data to the province installation. The province installation now contains its data plus the data from the district and the village, even thought the province did not import the data file from the village. The data from the village was contained within the export file from the district. Likewise, when the province installation exports its data to the country installation, the country installation will contain all of the data from the province, district, and village. Any installation that the country installation exports its data to will also contain the data from the village, district, and province. Eventually, data from each installation will migrate to every other installation in the country.

# Master and Slave Installations

All DDMS installations fall into one of two categories: master or slave. In order to avoid unrecoverable synchronization conflicts, certain actions, such as modification of the Geo Hierarchy, are permitted only on master installations. Additionally, data cannot be shared between two master installations, so it is critical that each deployment contains only one master. No more than one master installation may exist within any given country. Otherwise, non-recoverable errors can occur during the synchronization process. Such errors will permanently prevent an installation from participating in synchronization again.

# Export Sequence Numbers

A transaction is an atomic set of creates, updates, and deletes on one or more objects that collectively implement an action in DDMS. Data is exported in segments of transactions. When a transaction is exported it is assigned an export sequence number. This number is used to ensure that data is imported in the correct order and is complete. That is to say, the import node has all of the required data from the export node for import data to not be corrupt. When a node is importing data it cannot have any gaps in the export sequence number. If there is a gap in the sequence number, then the import node must contact the export node and retrieve the transactions for its missing sequence numbers.

Given the previous example where a village installation exports its data to the district installation, the exported data file contains transactions with export sequence numbers one, two, and three. Later, the district installation gets another export from the village. However, this new export contains transactions with export sequence numbers five, six, and seven. The district installation will not be able to import the village's second data file since it is missing the transaction with export sequence number four. The village must produce a new data file that contains this missing transaction. However, it is possible to import a data file that contains transactions with sequence numbers that have already been imported. Such transactions are simply ignored. Importing the same data file more than once will have no effect.

# Site Masters

During the install process each install is provided a unique identifier. This identifier is used to associate all data with the install that was responsible for creating them. It is extremely important that installs never share the same unique identifier. If duplicate identifiers exist, then DDMS will not be able to track where data originated. In such a situation, synchronization of data will behave incorrectly and may corrupt data to a point beyond recovery. Additionally, in order to prevent a large number of data conflicts, DDMS restricts modification of data to the install that created them. For example, a spray team created at the village install cannot be modified by the district install. Rather, it can only be edited at the village install. However, the synchronization resolver allows the user to by-pass this restriction. As such, in a worst-case scenario using the synchronization resolver, the village install would be able to modify data that originated from the district install. Note that when an install modifies data that originated from a different machine, it can cause a divergence of data when that data is propagated to other installs.

Basic CRUD operations

Data in DDMS is constructed primarily of Types and Relationships. An attribute is simply a value. Examples of attributes include *username*, *collection ID*, or *gender*. Attributes are discussed in detail later in this document. A type is a collection of attributes that have been grouped semantically to represent something in the system. For example, the *Person* type contains attributes for *first name*, *last name*, *date of birth*, and *sex*. A type can have multiple instances, each with different values. For example, one instance of Person could be John Doe, Male, born January 3, 1970, while another instance could be Mary Smith, Female, born May 5, 1973. Each type contains a special attribute called *Key Name*. Key Names must be unique, and typically conform to a pattern, though the pattern is different for each type. When editing a key name, follow the pattern seen in other instances of the type. Reference attributes, relationship trees, and several other widgets in the resolver display the key name of the instance. Relationships are designed to connect instances of two different data types. For example, the *In Team* relationship connects a *Spray Member* to a *Spray Team*. DDMS utilizes two perspectives on data: the view model and the data model.

Like most modern web applications, DDMS uses a database to store and retrieve all data. The synchronization resolver and web application use the same database. The data model is designed to maximize performance and minimize repetition whereas the view model is designed for presentation to the user via the web application. Consequently, the data model differs from the view model. An example of the view model is the Spray Team screen, which shows data from several different data types: *Spray Leader*, *Spray Member*, and *Geo Entity*, in addition to the *In Team* relationship that links members to the current *Team*.

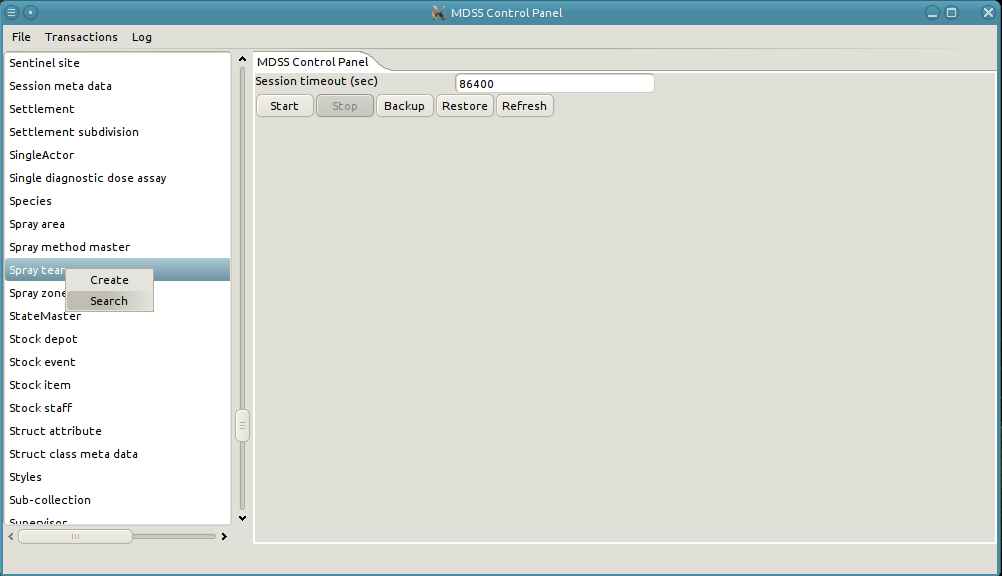
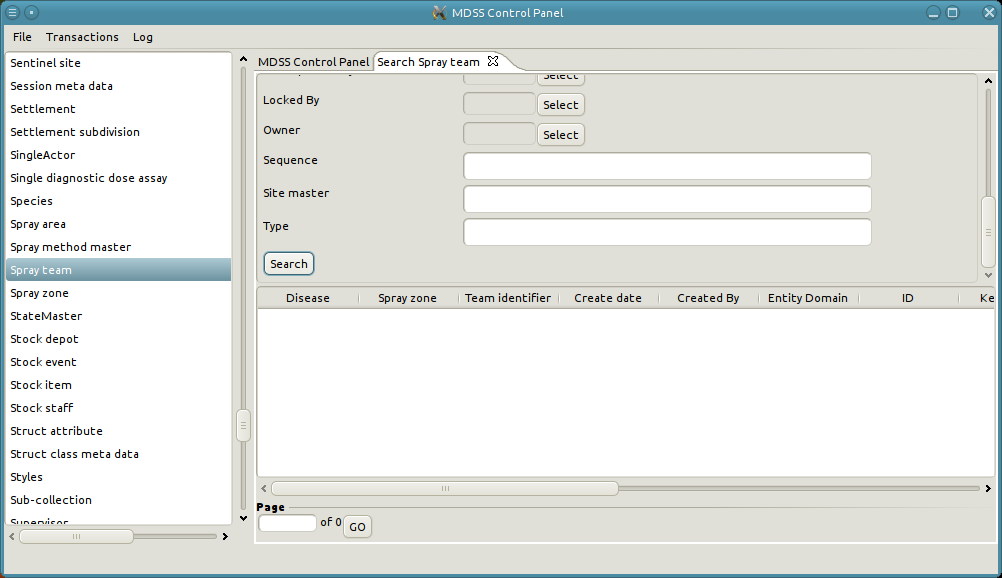
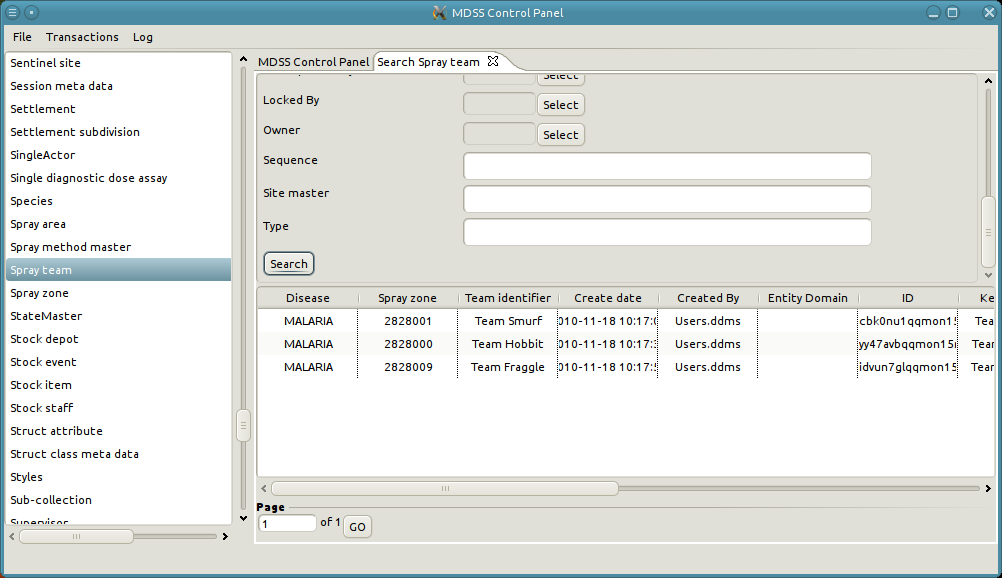
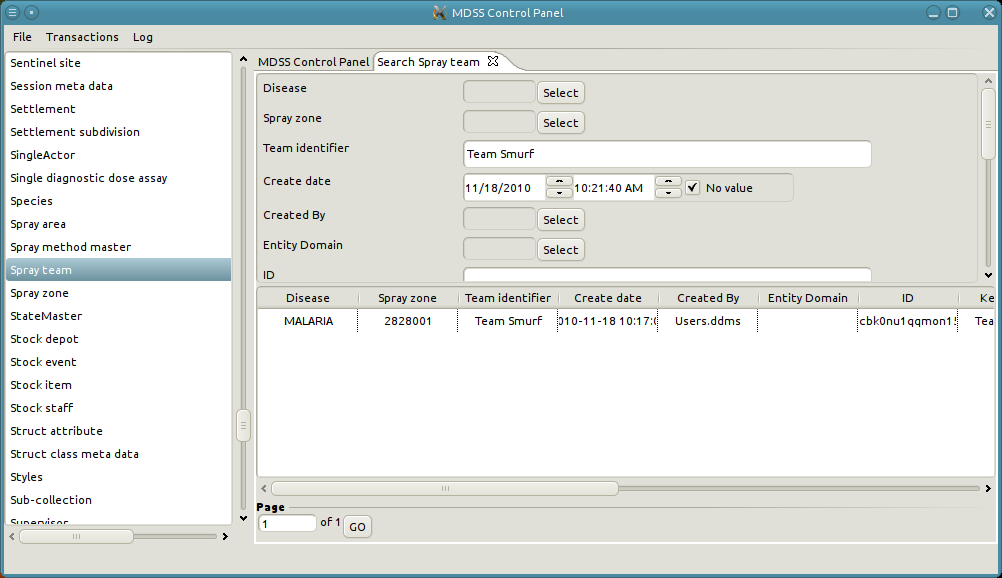
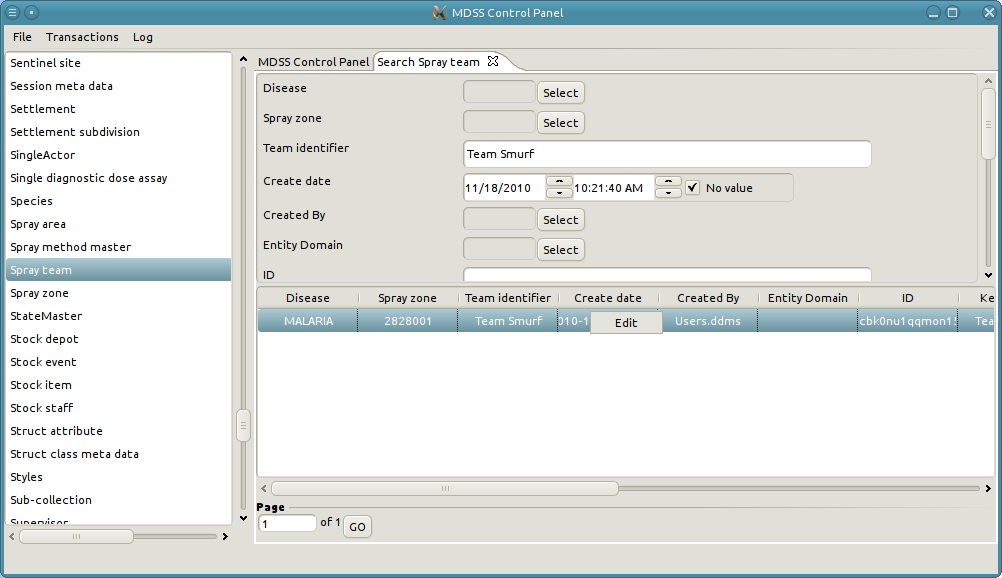
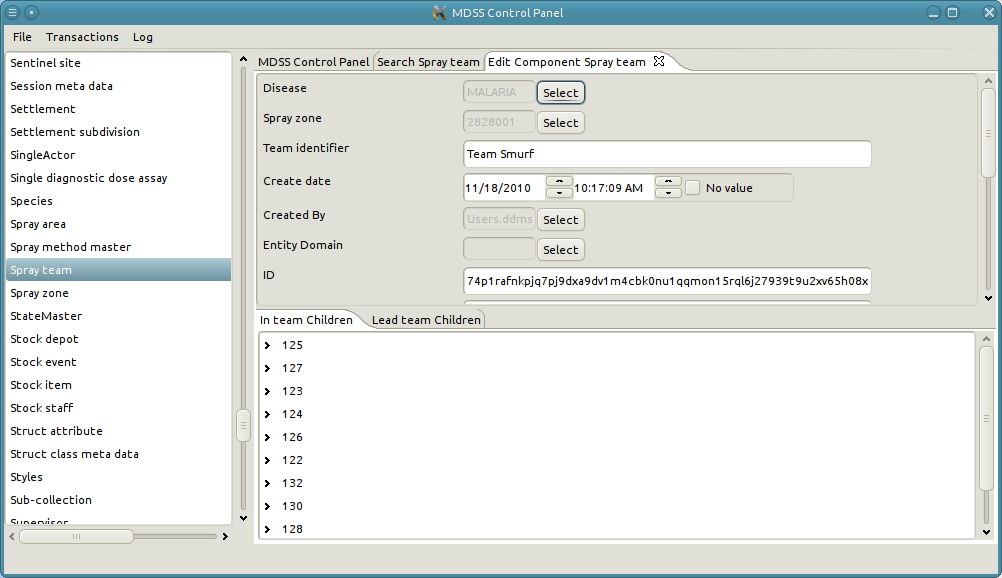
The synchronization resolver uses the data model to directly modify objects in the database. While this enables it to resolve all conflicts, it also presents the potential for data corruption. For example, if a *Spray Team* and a *Spray Member* are created through the resolver, but are not linked with an In Team relationship, then the member is not considered a part of that team. While this example is relatively simple, DDMS contains many complex structures that could crash the web application if data were imported through the synchronization resolver improperly. Thus a strong understanding of the data model of DDMS (not just the view!) is necessary to avoid data corruption.

# View existing data

The basic use case for viewing data in the system is as follows:

1. The left panel lists all types in the system. Note the list of data types does not include relationships, which are covered extensively in the next section. Double click a data type to search for records of that type. Alternatively, right-click on the data type and select "Search." A new tab opens with a search form for the selected type
2. Fill in search criteria as needed
3. Click "Search" at the bottom of the form. The results do not auto-update, so as new criteria is added to the form, "Search" must be clicked again before new results will appear.
4. Search results appear in the paginated table below the form. The results can be sorted on non-reference attributes by clicking the column header.
5. [Optional] Access additional pages of search results with the input field below the results table
6. Double click a row of the result table to view the object

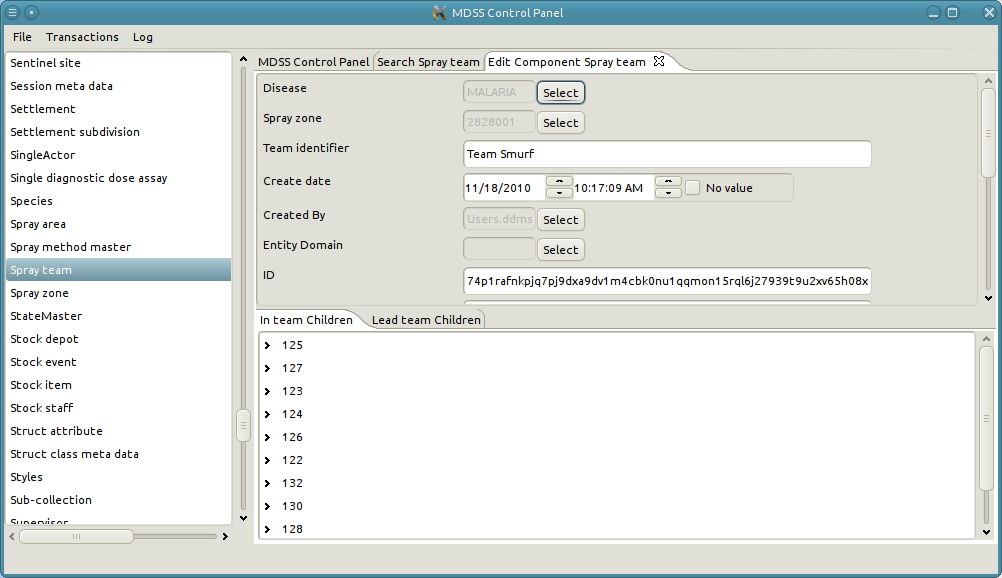
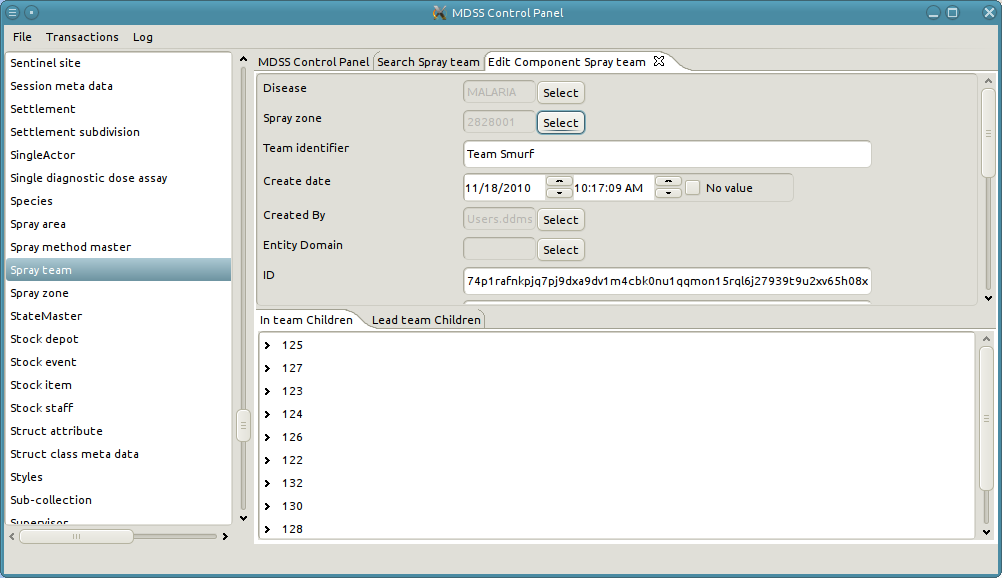
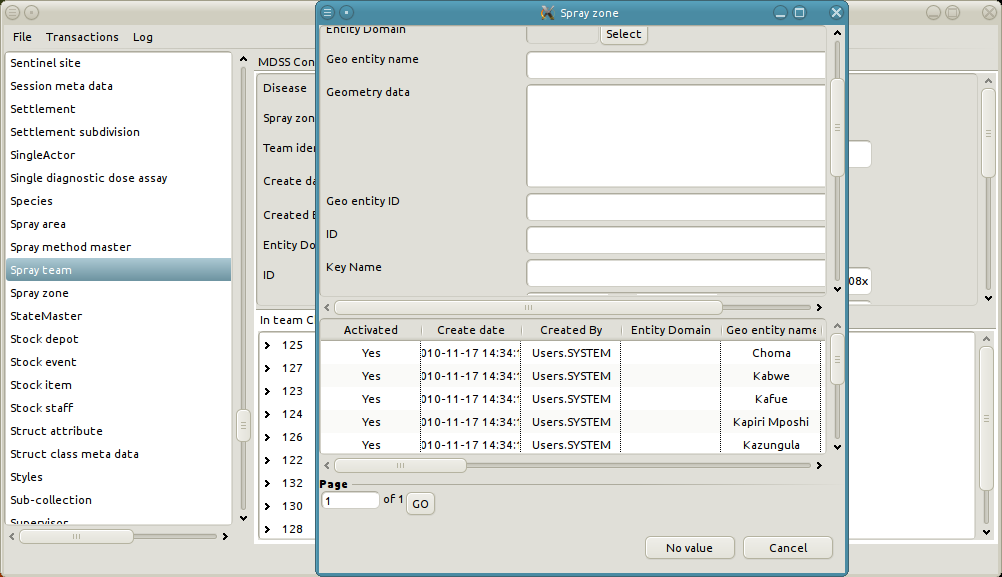
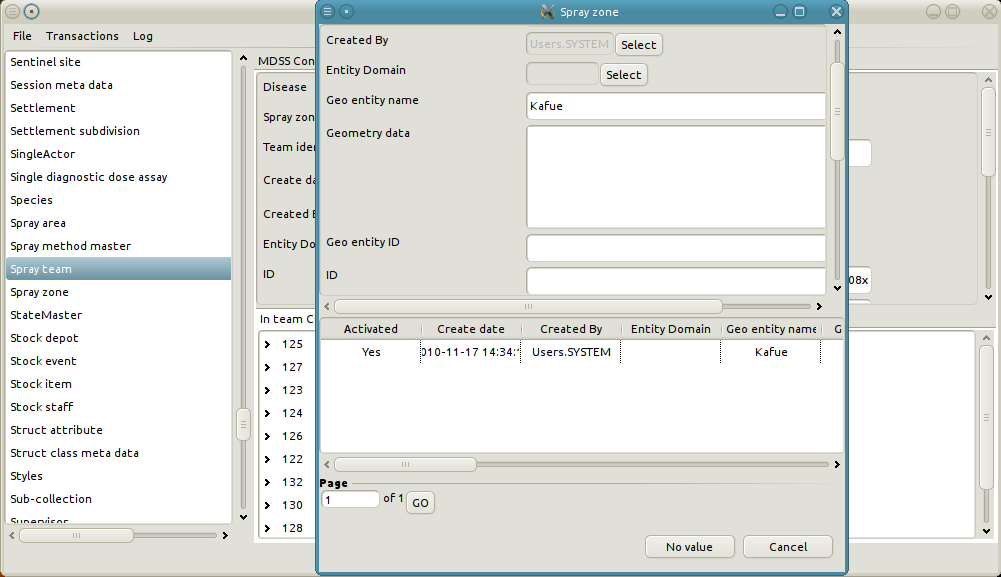
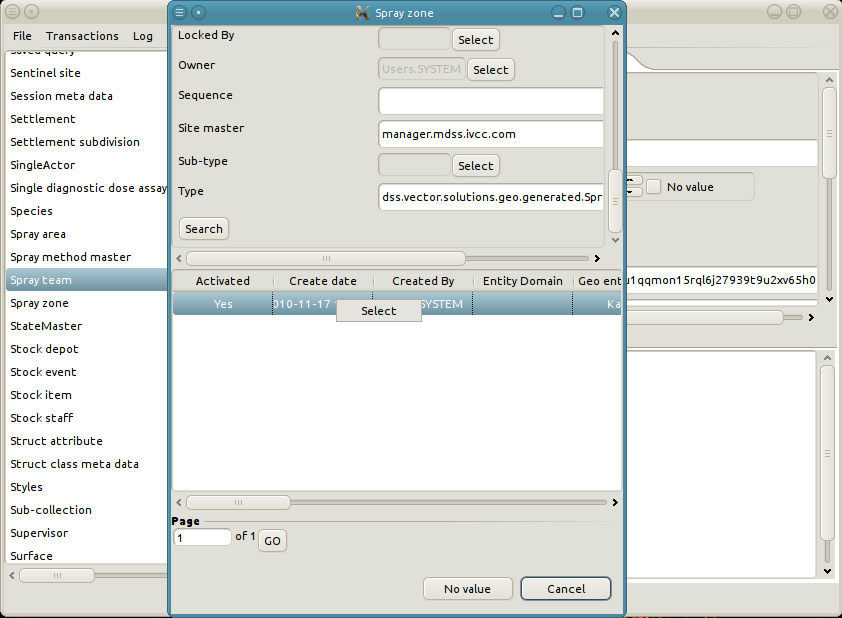
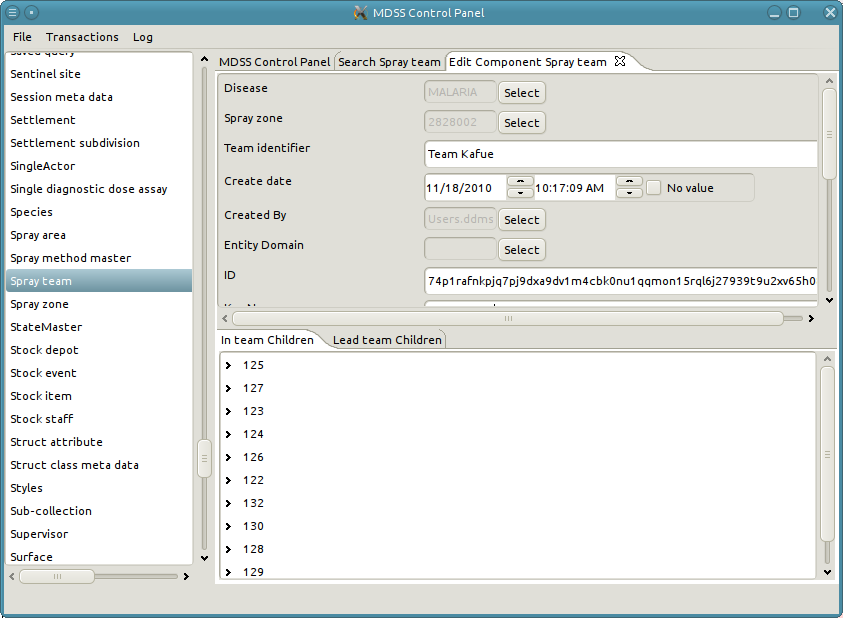
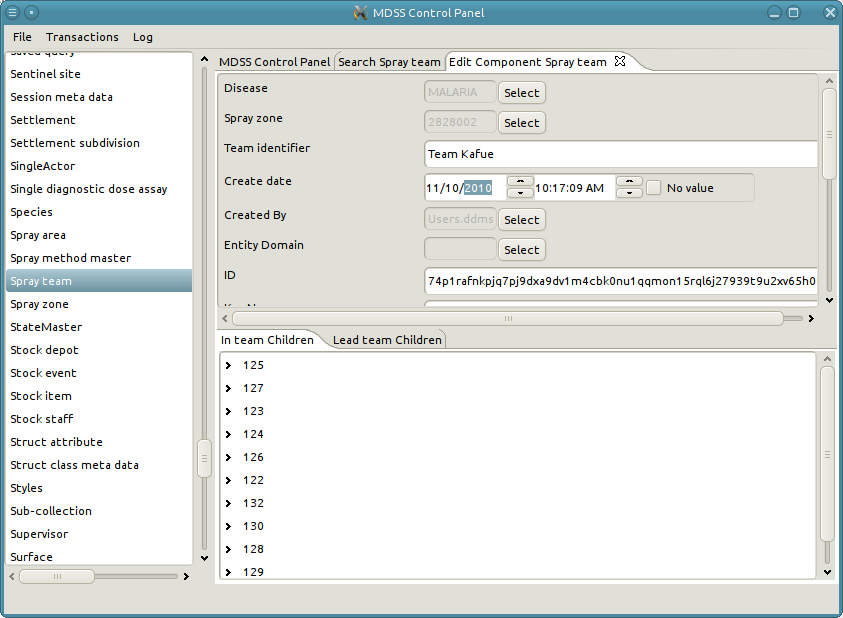
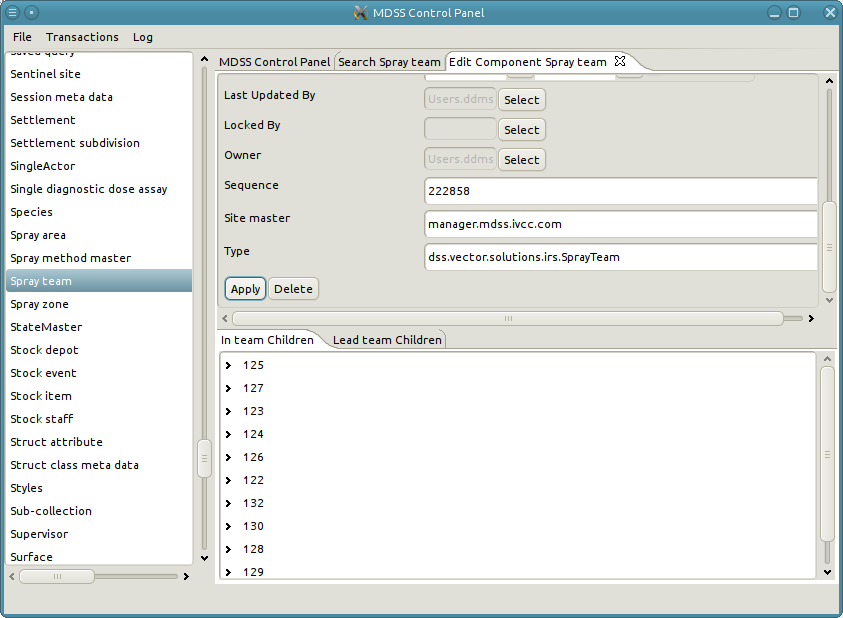
# Example of searching for and viewing a spray team

1. Select Spray Team from the list of types to search. 
2. A new tab is opened with the search form for spray teams.
3. Click on search with an empty form to see all of the spray teams in the system. If the results span multiple pages, more results can be displayed by changing the page number and clicking on "Go".
4. Optionally, results can be restricted by adding additional criteria to the search form. For instance, restricting the Team identifier to "Team Smurf"
5. View the details of "Team Smurf" by right-clicking on its row and select edit or by double-clicking on its row.
6. This brings up a new tab with the details of the spray team.

# Edit existing data

Editing existing data is the most frequent action in the synchronization resolver. After searching for and locating an instance, the resolver can be used to change attribute values on that instance. Refer to the Attributes section for a detailed explanation of different attributes and how to edit them.

# Example of modifying existing data

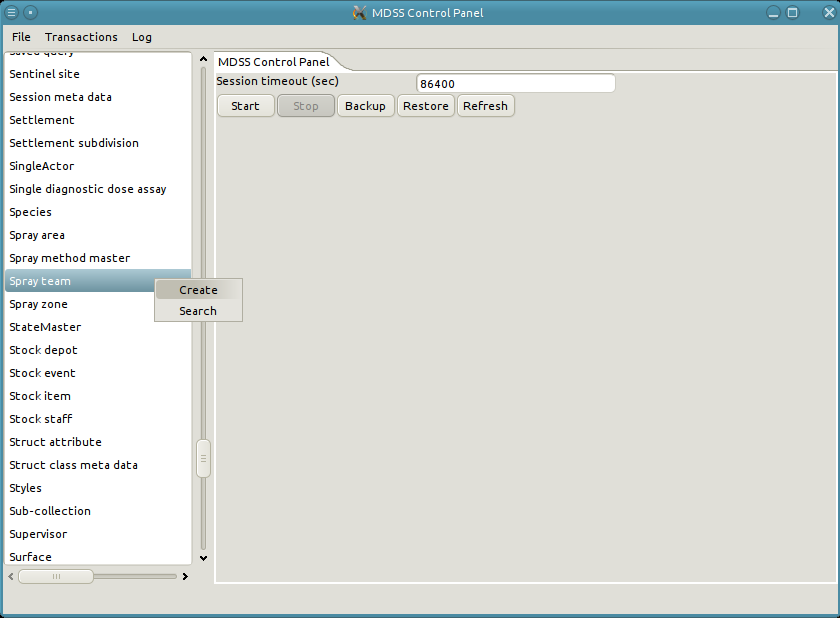
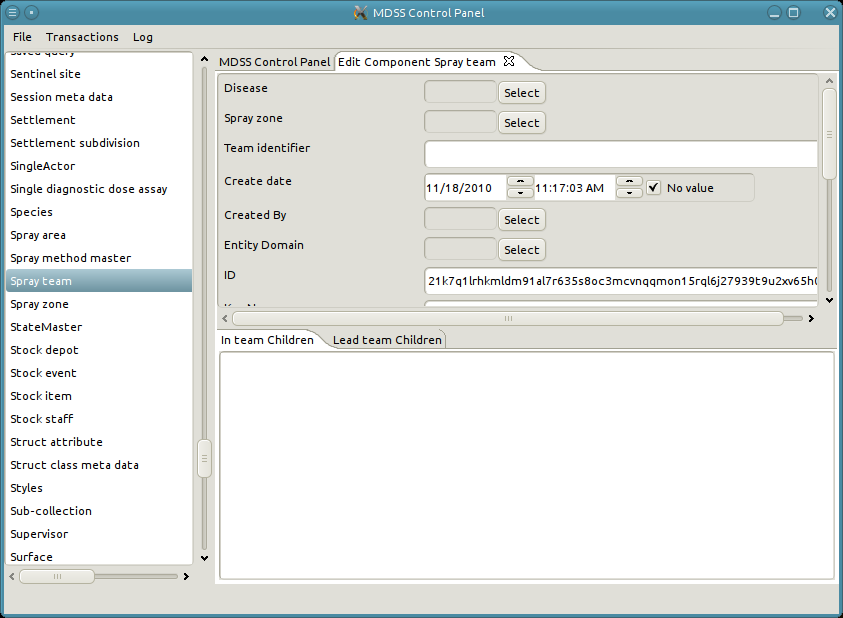
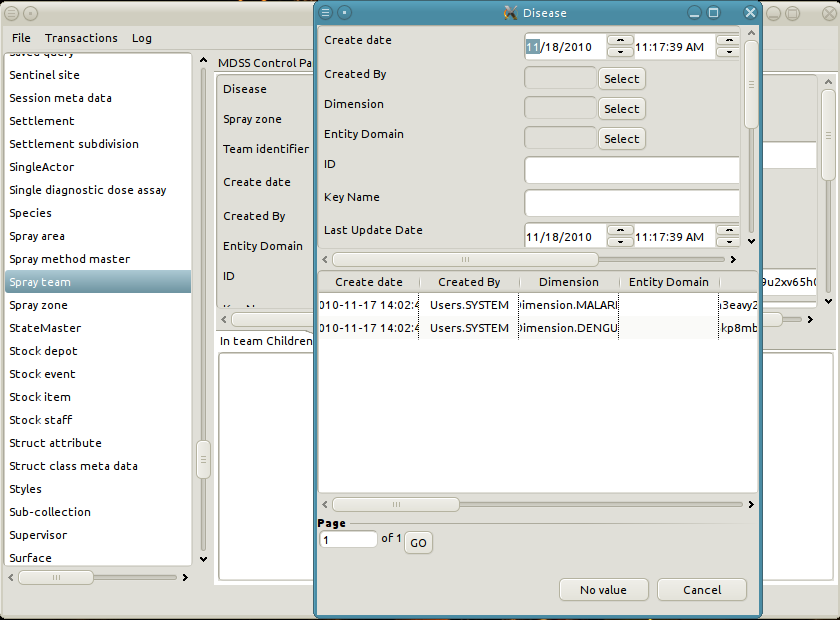
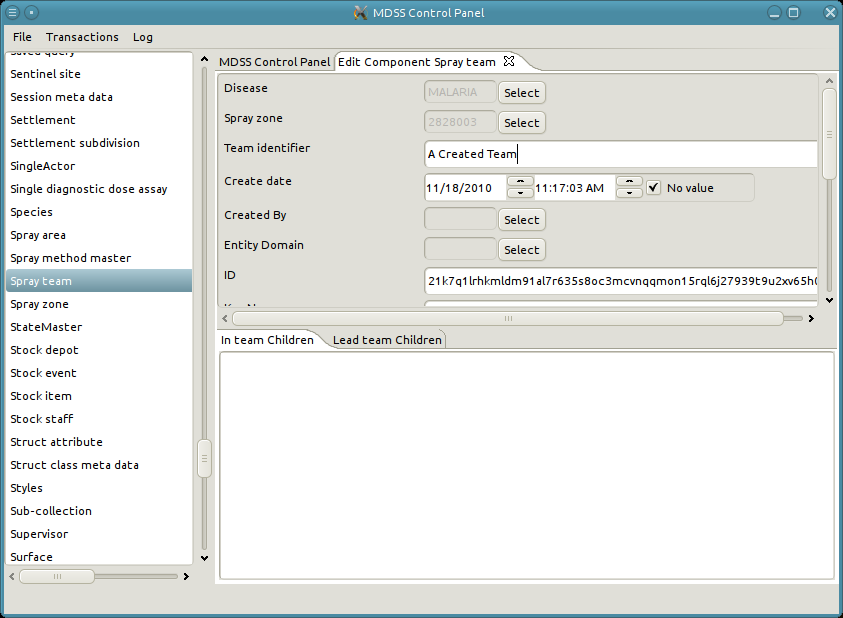
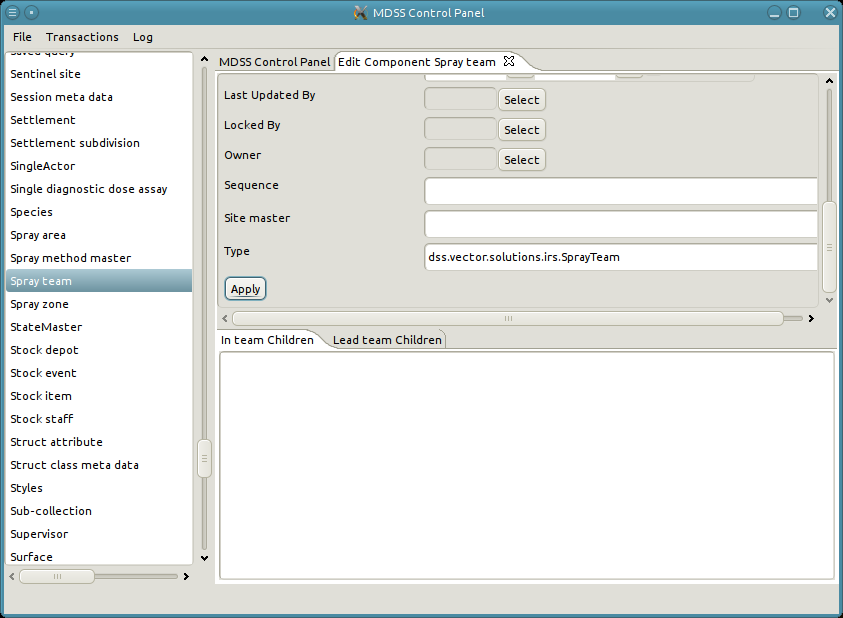
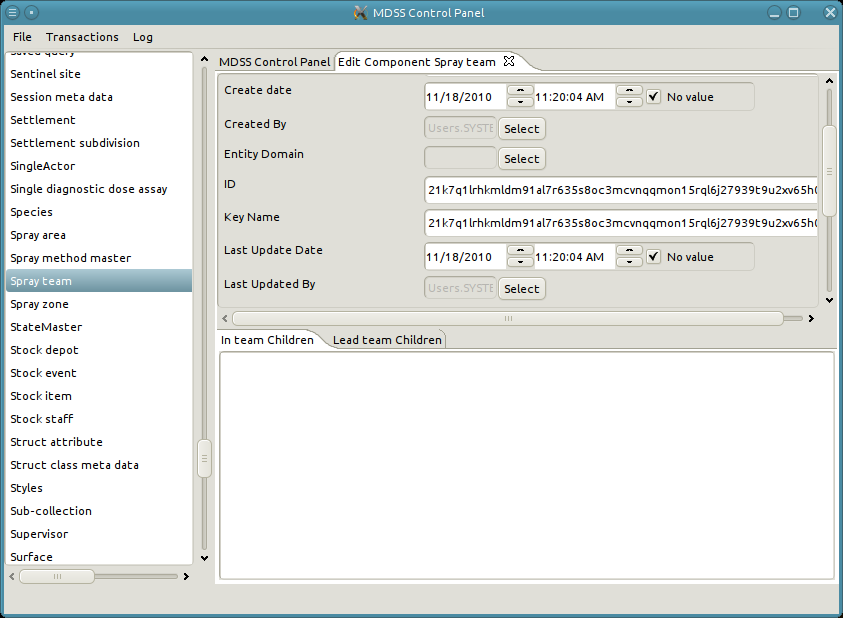
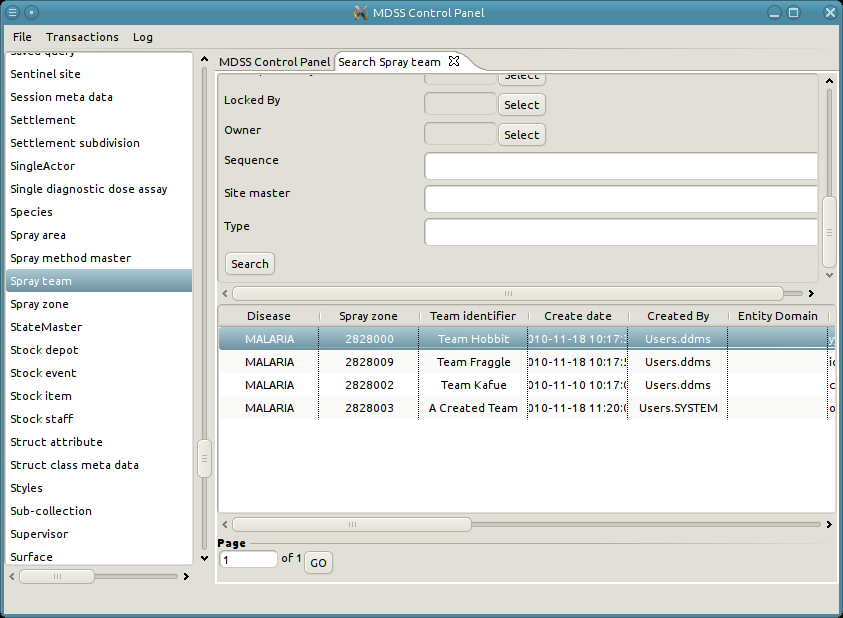
1. Continue from the previous example of viewing the spray team "Team Smurf"
2. Modify the spray zone of the spray team. Start by clicking on the "Select" button next to the Spray zone input field.
3. A pop-up opens which can search for and select a spray zone. If the Spray zone field already has a value then the pop-up will initially appear with data from the selected spray zone. Note that the user can decide to not change the value by selecting "Cancel," or the user can elect to remove the current value by selecting "No value."
4. Remove some of the criteria in the form to get a broader result set. This can also be accomplished by selecting "No Value" and then hitting the Spray zone "Select" button again.
5. Similar to a search tab, list of possible spray zones can be refined by adding criteria to the form. For instance, change the entity name to “Kafue.” 
6. To select the "Kafue" spray zone right click on the row and choose "Select" or double click the row.
7. The Spray zone of the Spray team reflects the selection of the "Kafue" spray zone. Similarly, change the Team identifier of the Spray Team. In order to change the Team identifier simply type in the desired value in the Team identifier text field. 
8. Finally, modify the Create date of the Spray Team to 11/10/2010. 
9. Now that all desired values have been modified, click “Apply” to save the changes to the database. 
10. Search the spray teams to confirm that "Team Smurf" is now “Team Kafue"

# Create new data

Data creation through the synchronization resolver should be approached with tremendous caution. In most cases, the data model is complex, and mimicking the creation of data as seen through the web interface requires the creation of several connected instances (See the Introduction to Basic CRUD Operations for a description of the differences between the view and data models). Failure to correctly create and connect all of the necessary instances and object can result in data corruption or instability, but the option to create is still included as a last resort.

1. The left panel lists all data types in the system
2. Right-click the desired type from the list and select "Create"
3. A new tab opens with the create form for the selected data type
4. Fill in the form
5. Click “Apply”

# An Example of creating a new Spray team

1. Right click on Spray team in the left panel and select "Create".
2. A new tab opens with the form to create the new Spray team.
3. Set the Disease to malaria. 
4. Second, set the Spray zone to "Kapiri Mposhi".
5. Set the Team identifier to "A Created Team".
6. Finally, save the new Spray by clicking “Apply.”
7. Notice that some of the system attributes were automatically updated once the Spray team was applied. These attributes are used internally by the system for book keeping and to ensure data integrity. 
8. Search all of the Spray teams in the system to confirm the newly created Spray team.

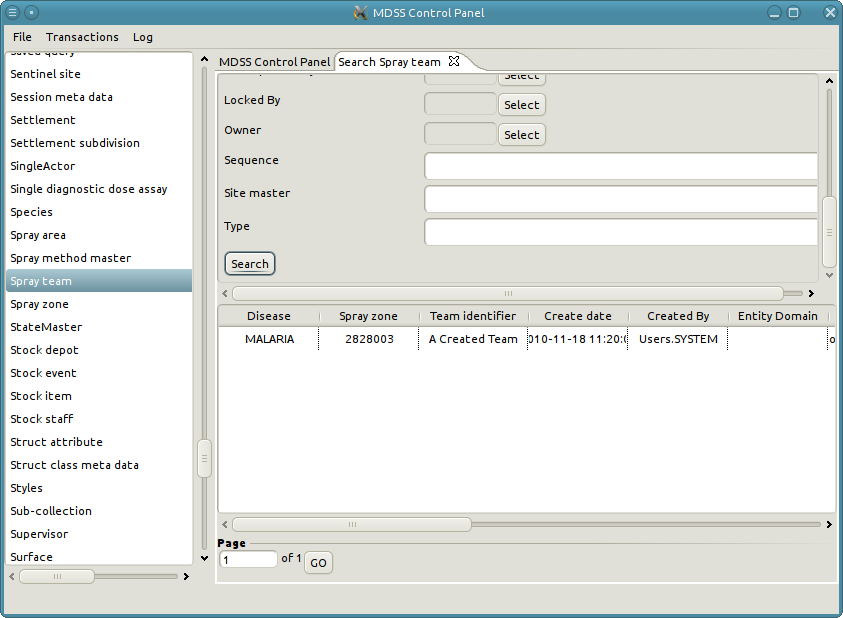
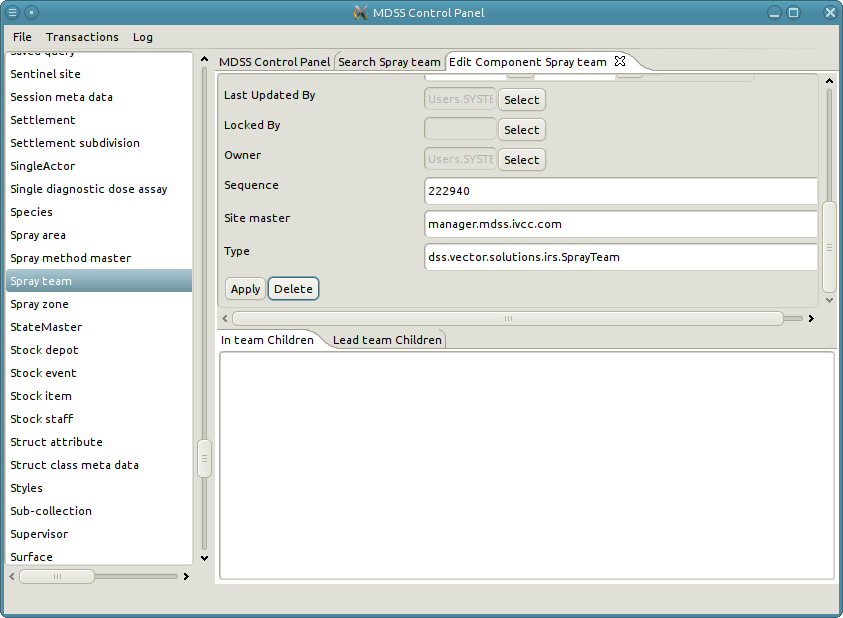
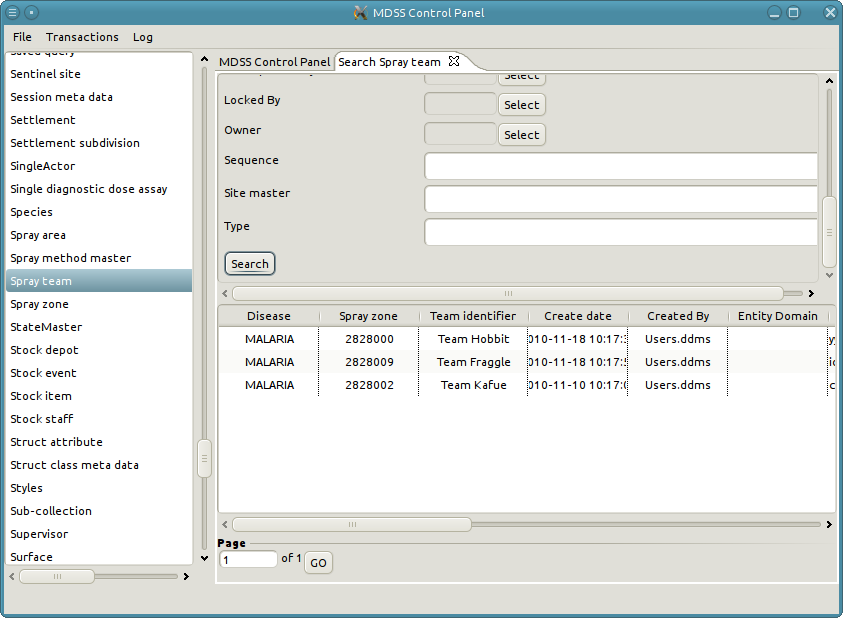
# Delete existing instance

Deleting objects is a highly dangerous operation. Many types are referenced by relationships or reference attributes, and deleting an instance can clear reference attributes or delete associated relationships as well. Conversely, an attempt to delete can fail if the instance is still referenced by a required reference attribute. Predicting all of the consequences of a delete operation is difficult, and often spans multiple data types and relationships. If at all possible, it is much safer to delete through the web interface rather than through the resolver.

1. The left panel lists all data types in the system
2. Double click the data type to search for records of that type. Alternatively, right-click and select "Search"
3. A new tab opens with a search form for the selected type
4. Fill in search criteria as needed
5. Click "Search" at the bottom of the form
6. Search results appear in the paginated table below the form
7. [Optional] Access additional pages of search results with the input field below the results table
8. Double click a row of the result table to view the object. Alternatively, right-click and select "Edit"
9. Click the "Delete" button next to "Apply" to delete the object.

# Example of deleting a Spray team

This example shows how to delete the Spray Team created in the previous example.

1. First search the Spray teams for "A Created Team".
2. Double click on the row to open up a new tab with the details of the Spray team. 
3. Click the "Delete" button.
4. Note when deleting the tab of the deleted Spray team automatically closes. Search Spray teams to verify that "A Created Team" is gone.

Relationships

A relationship associates two instances of data. The most common need for a relationship is one instance referencing many others. For instance, a Spray team has many Spray operators. As such, the data model could have attributes called “Operator 1,” “Operator 2,” and “Operator 3,” but that approach limits the number of operators on a team to 3. By using a relationship instead of attributes, the team can have exactly as many operators as it needs - no more, no less. Additionally, like objects, relationships can have their own attributes. For example, to track when a Spray operator was added to a Spray team, an attribute can be added to the relationship object.

All relationships are directed between their two participants; participants can be thought of as source and sink nodes, or parent and child nodes. In the example where a Spray team has many Spray operators, the source/parent node is the Spray team and the sink/child node is the spray operator.

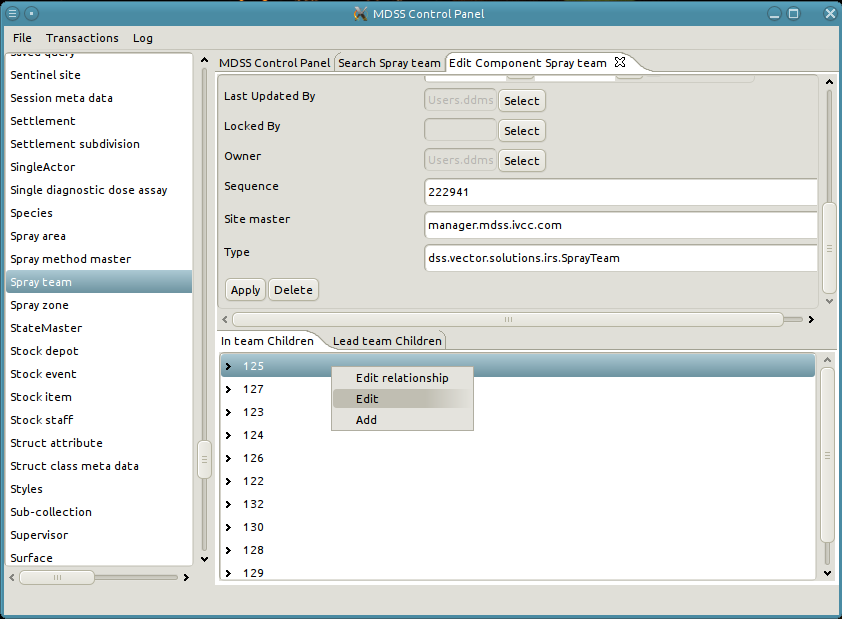
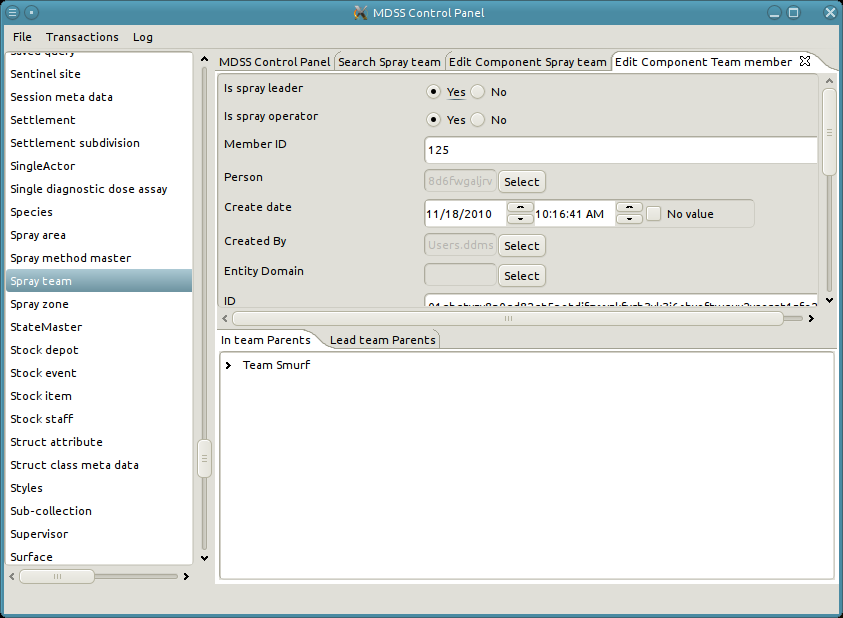
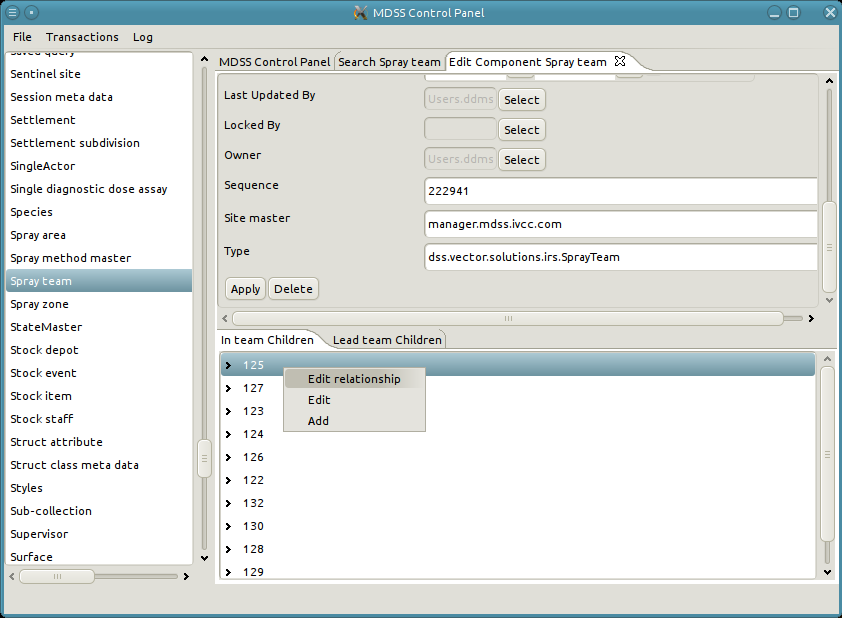
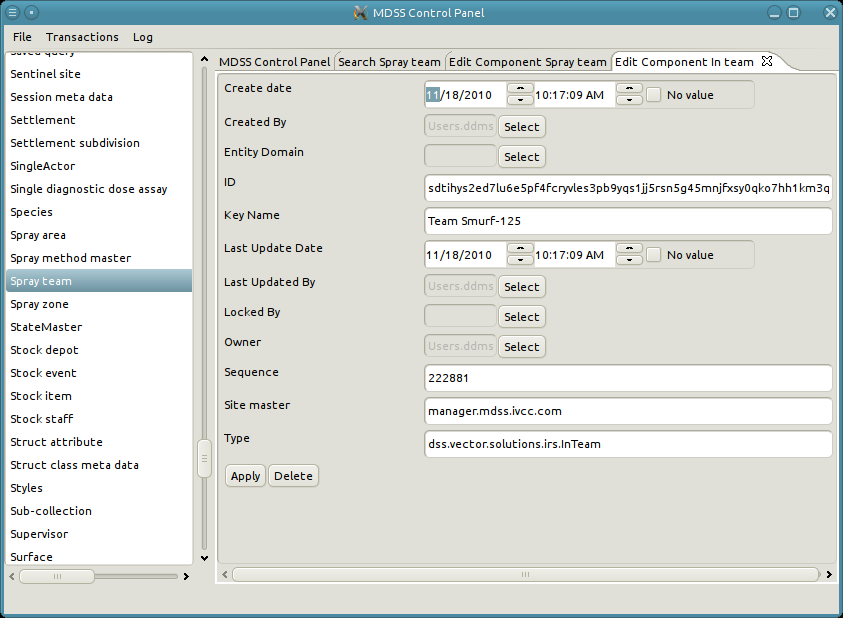
# Viewing relationships

Relationships are not searched for directly; rather the participating objects are searched for and the relationships they participate in are shown. This approach has the advantage of showing all the different relationship types an instance participates in.

1. Search for the participating instance
2. The bottom panel contains tabs for each type of relationship the object participates in. The tab also indicates the direction (parent or child) of the relationship.
3. Select the tab with the correction relationship and direction
4. The panel contains a tree structure representing the objects on the other end of the relationship. Each object is represented by its key.
5. If a relationship is cyclical such that the parent and child are of the same type, the tree can be expanded indefinitely, to follow the chain of relationships.
6. To view the details of the participating object right-click on the node and select "Edit"
7. To view the details of the relationship right-click on the node and select "Edit Relationship"

# Example of viewing the relationships

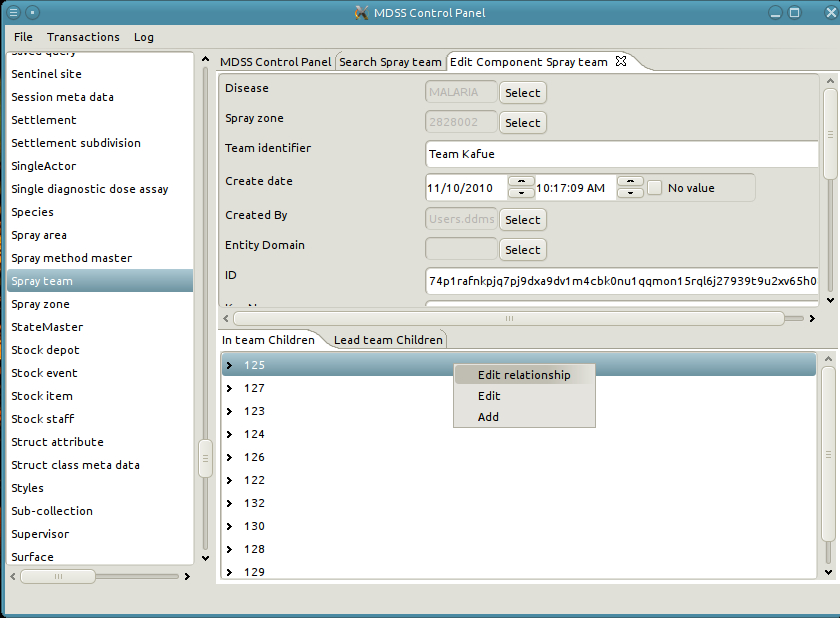
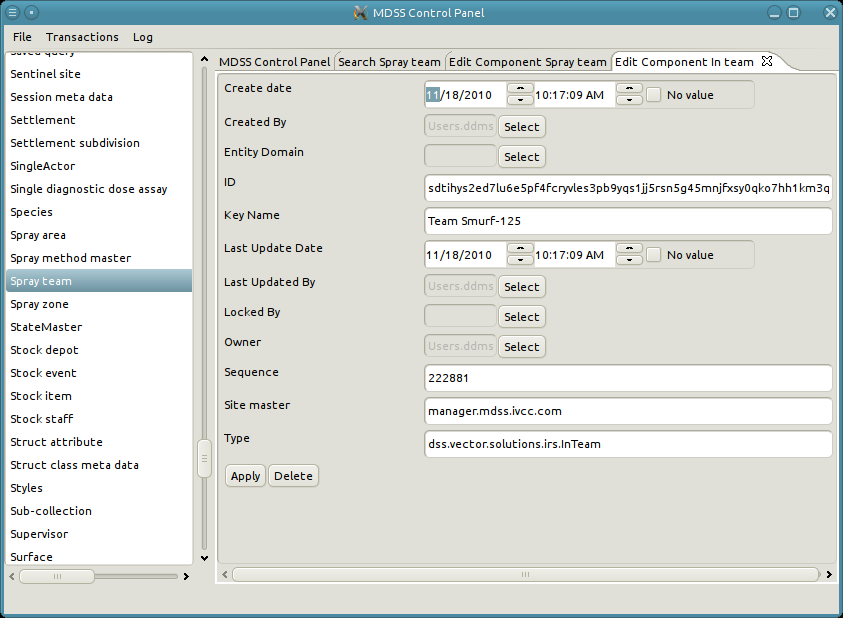
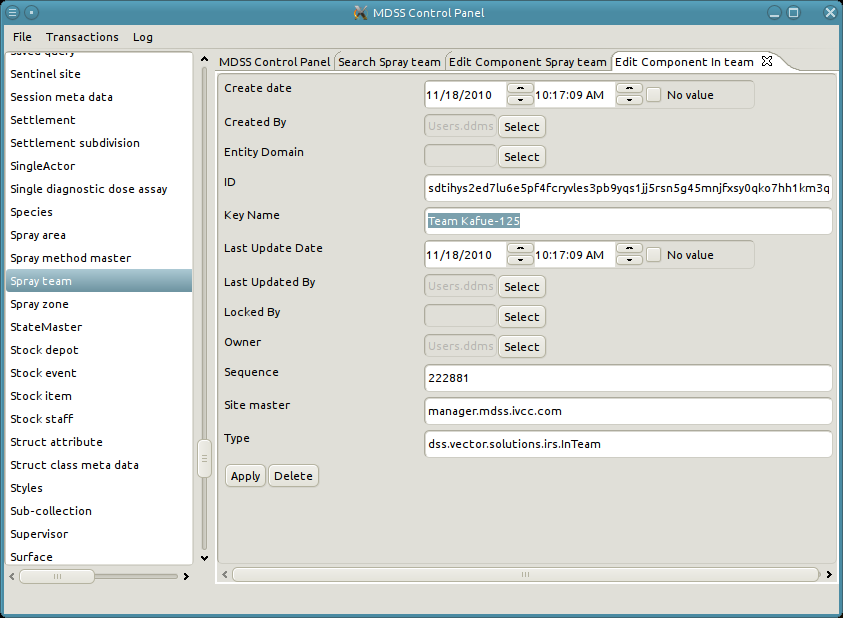
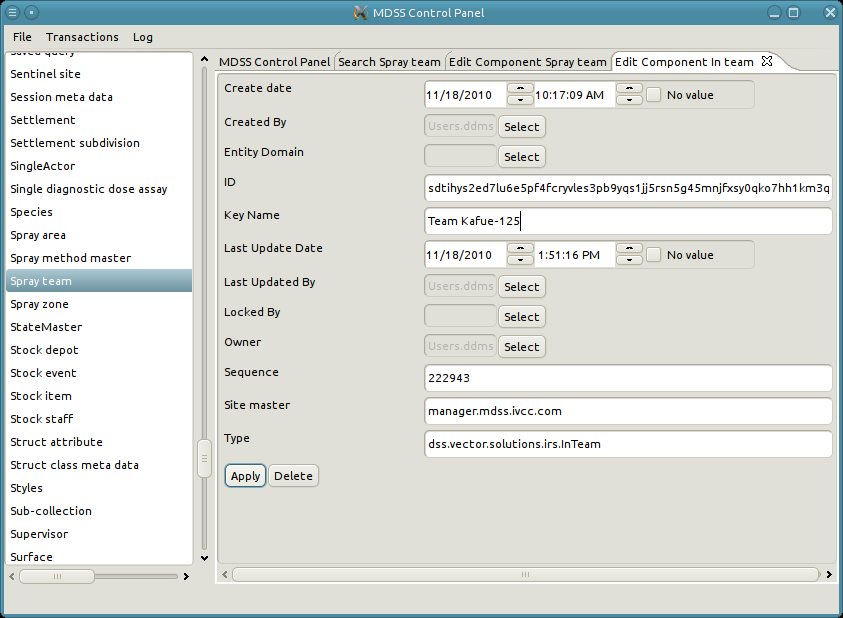
In the previous example the Spray team "Team Smurf" has many Spray operators. This is modeled through the "In team" relationship. Notice under the "In team Children" tab is a list of numbers. These numbers are the keys for the Spray operator objects.

1. Examine the participating object by right-clicking on its node and selecting "Edit"****
2. A new tab opens with the details of the Team member object. 
3. To examine the relationship object itself, click the "Edit relationship" option. 
4. A new tab opens with the details of the relationship object. 

# Edit an existing relationship

Editing a relationship is basically the same as editing an object. The primary difference is finding the relationship object in which to edit. See "Viewing a relationship" for directions on viewing relationships. One important fact to note is that it is impossible to change the parent or child or a relationship. If a relationship is created with the wrong parent or child, the only recourse is to delete the relationship object, create a new one with the correct parent and child, and copy the values from the deleted relationship.

# An example of editing a relationship

1. Search for the Spray team "Team Kafue," right click Team Member 125 of the *In Team* relationship, and click "Edit relationship". 
2. A new tab opens for the relationship.
3. Change the Key Name to "Team Kafue-125".
4. Apply the relationship. 

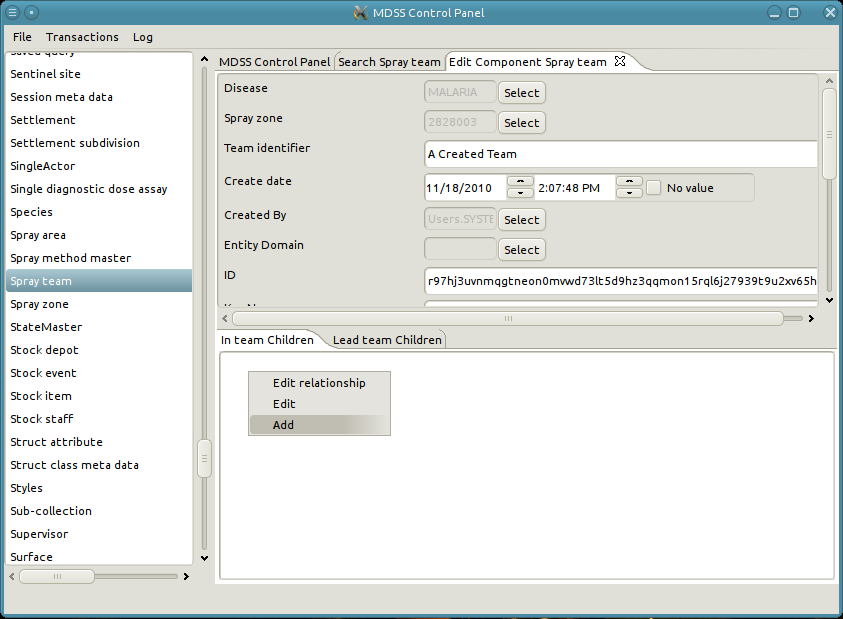
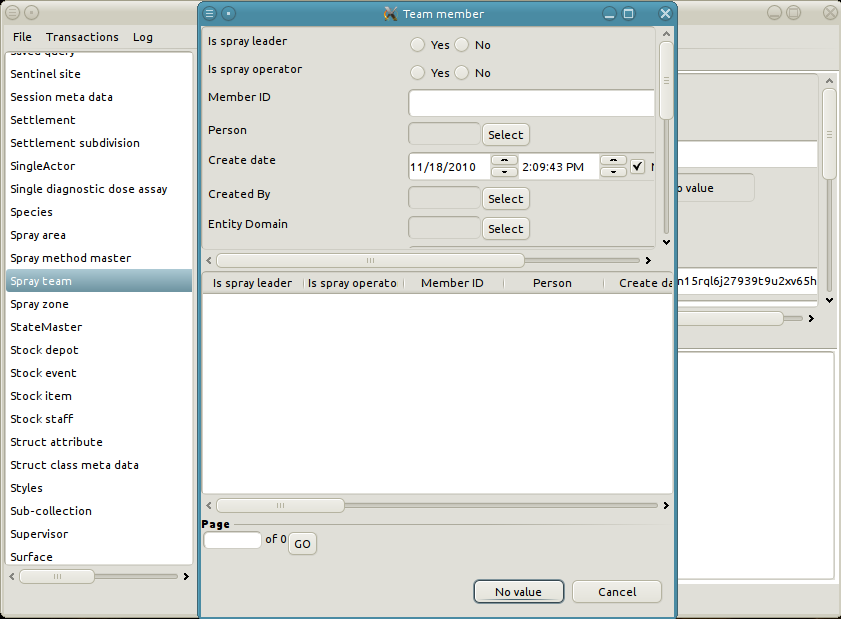
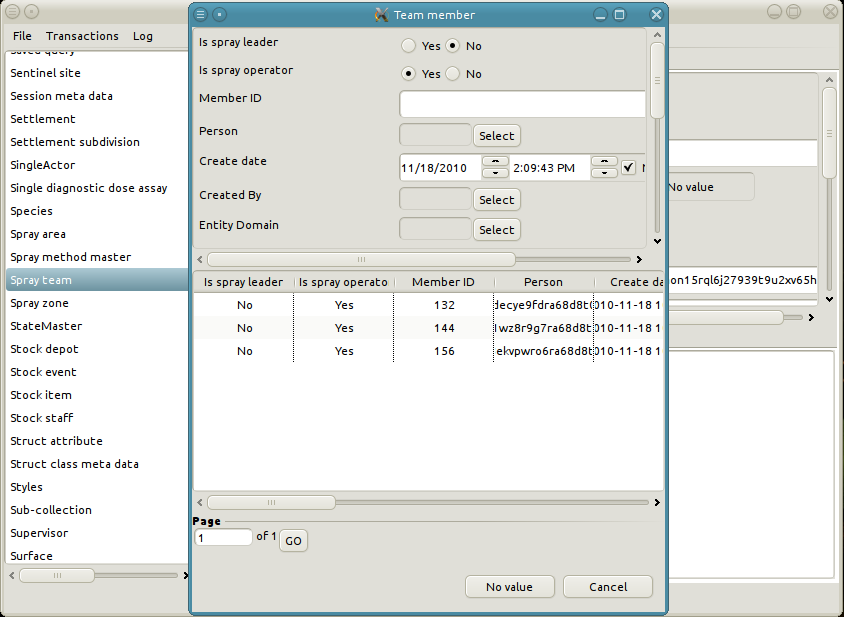
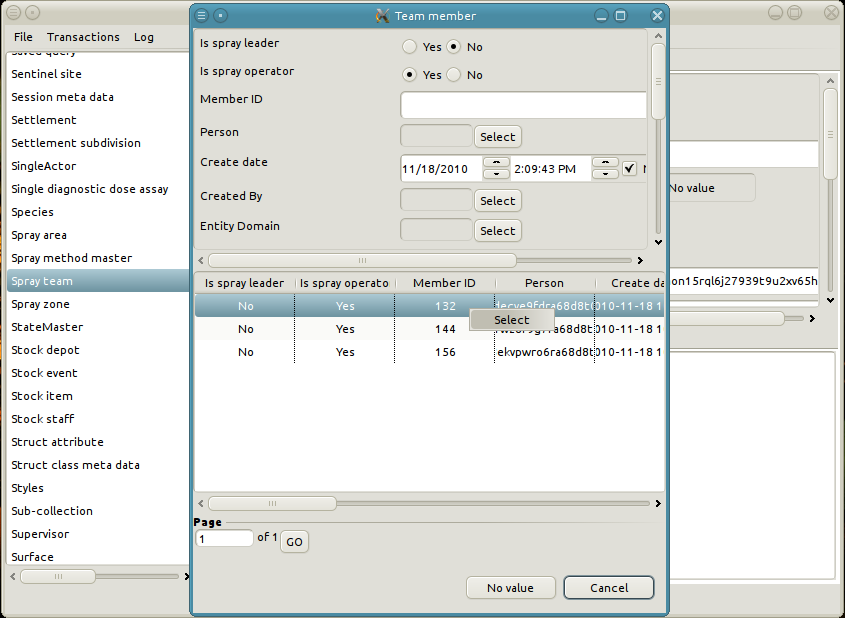
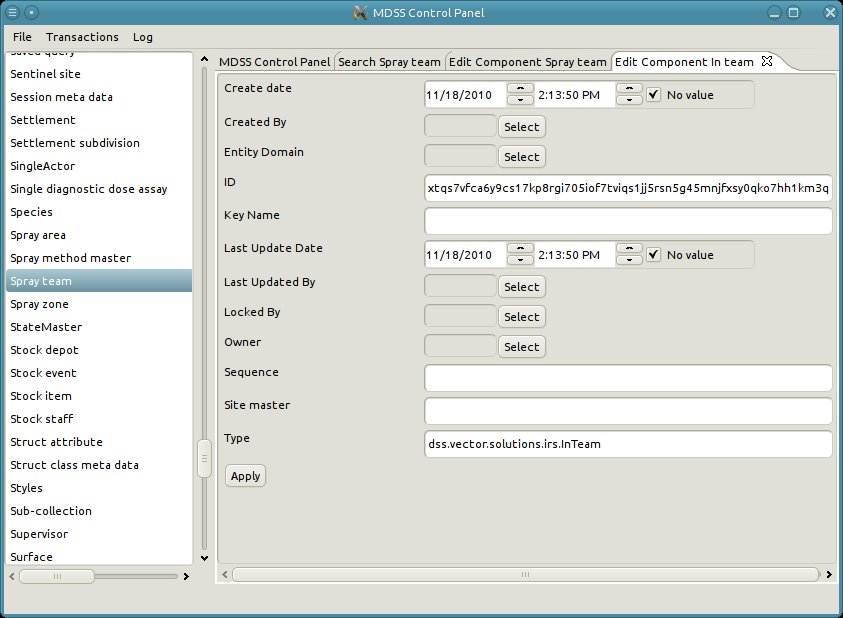
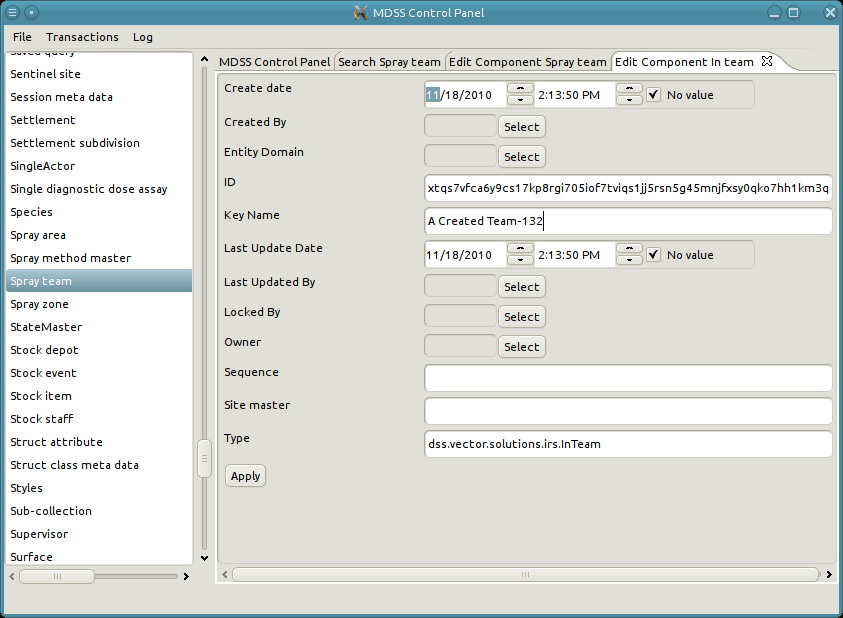
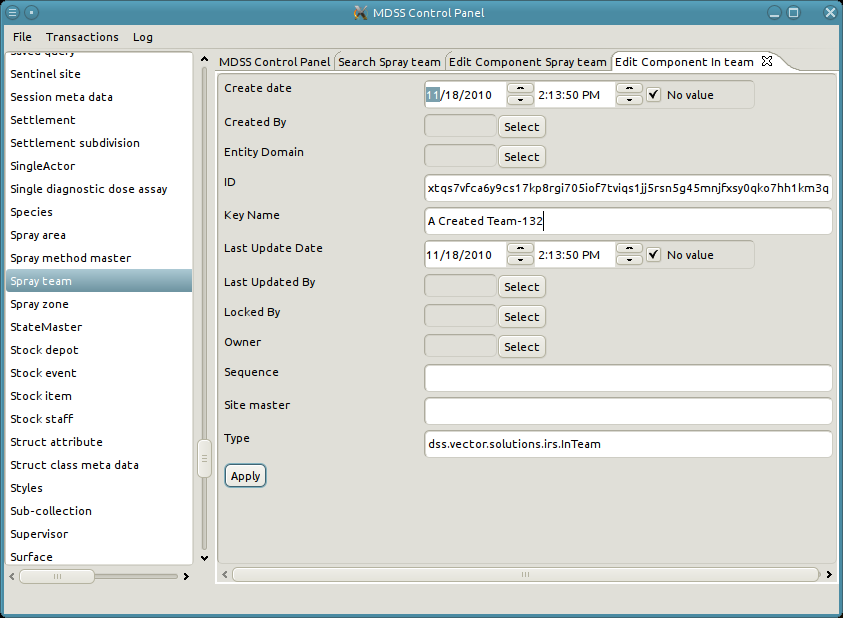
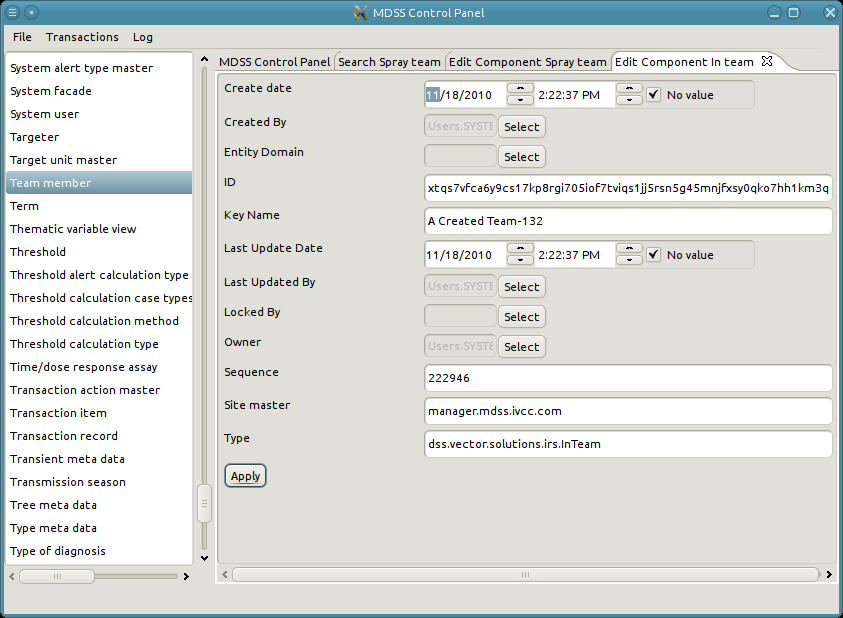
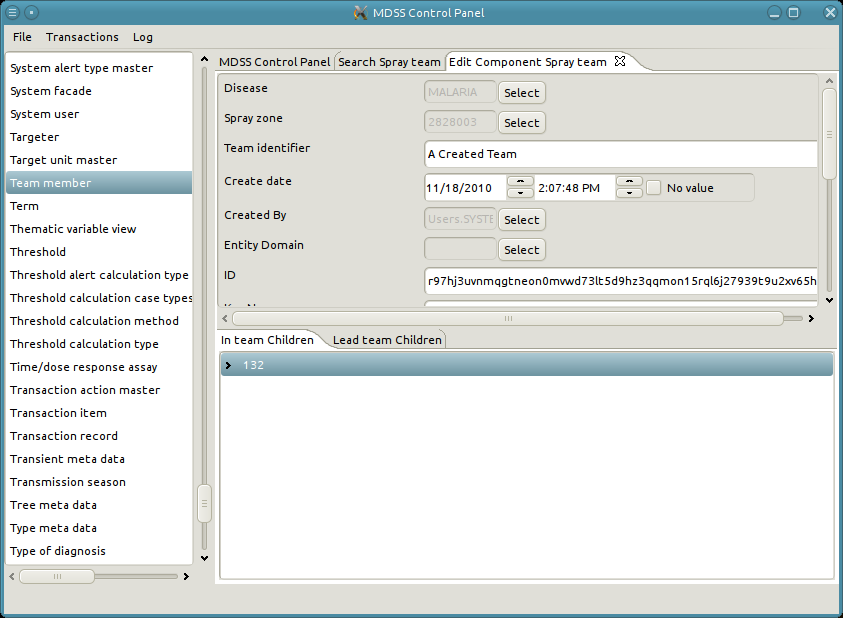
### Add a new relationship

Adding a relationship is accomplished in the same panel as viewing relationships. See "Viewing a relationship" for details on viewing relationships.

1. Right-click anywhere in the relationship panel select "Add a relationship"
2. A pop-up appears to search for and select the other participating object in the relationship (See Reference Attribute for more info)
3. Once the other participating object has been selected a new tab appears for editing the relationship.
4. Edit attribute values as desired
5. Click "Apply" to save the relationship.

# An example of adding a new relationship

This example shows the addition of a Spray Operator to the Spray team “A Created Team” created in a previous example.

1. Select "Add a relationship" from the “In Team” relationship panel. 
2. A pop-up opens to search for a Team Member.
3. Set the form to search for Team members who are Spray operators and not Spray leaders.
4. Double click to select Team member "132." Alternatively, right-click and choose “Select.”
5. A new tab opens with the form for editing the relationship object.
6. Change the key of the relationship to "A Created Team - 132.”
7. Click apply to save the relationship.
8. Some fields will be auto-populated when the relationship is applied
9. View the Spray team "A Created Team" to see the new relationship with Team member "132"

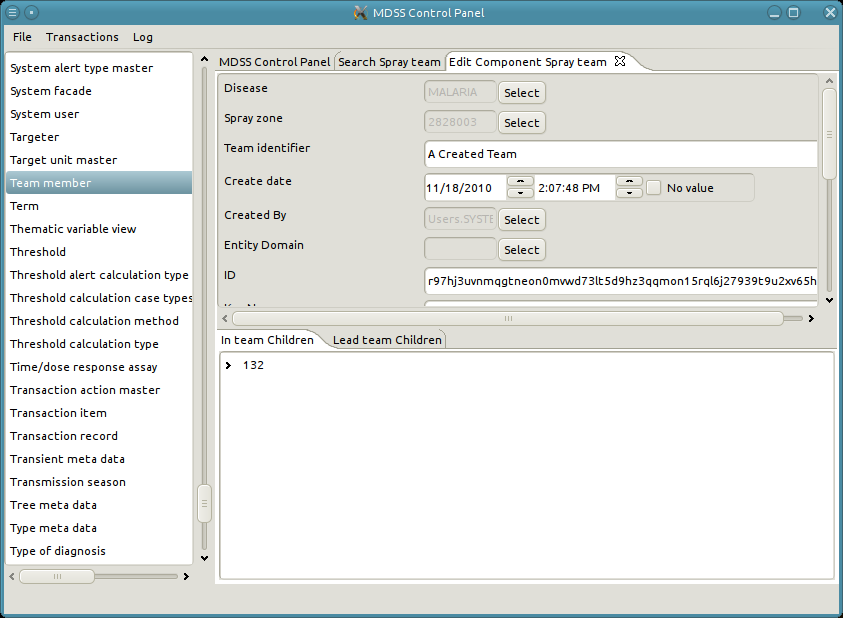
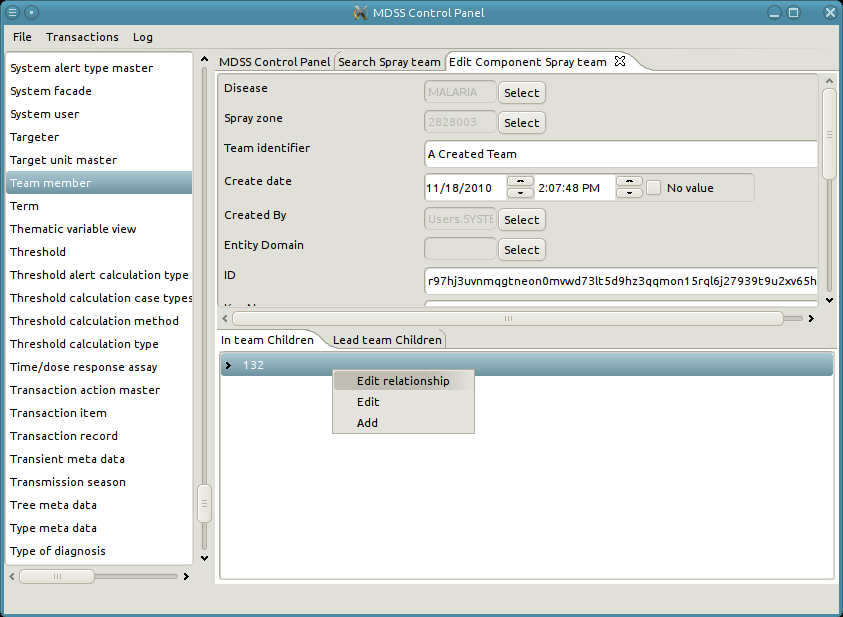
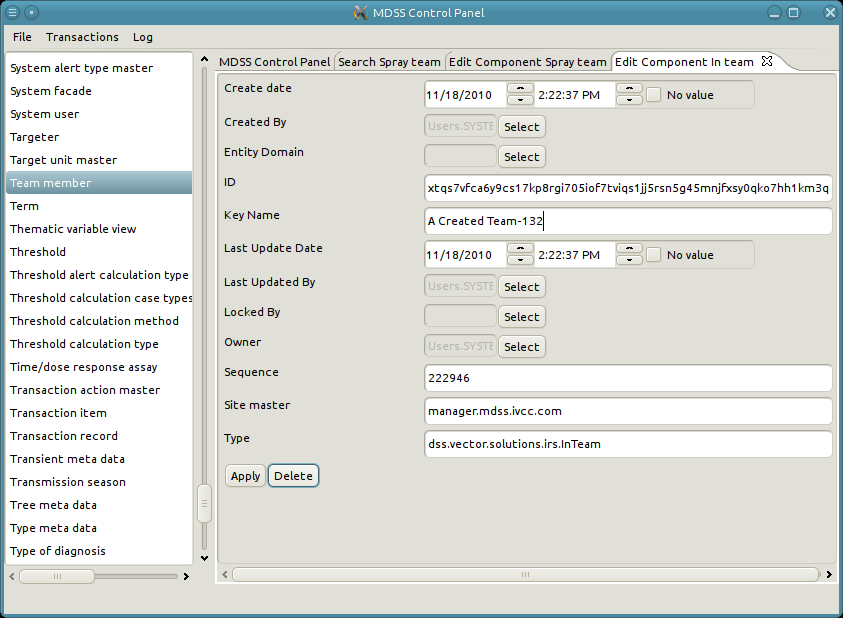
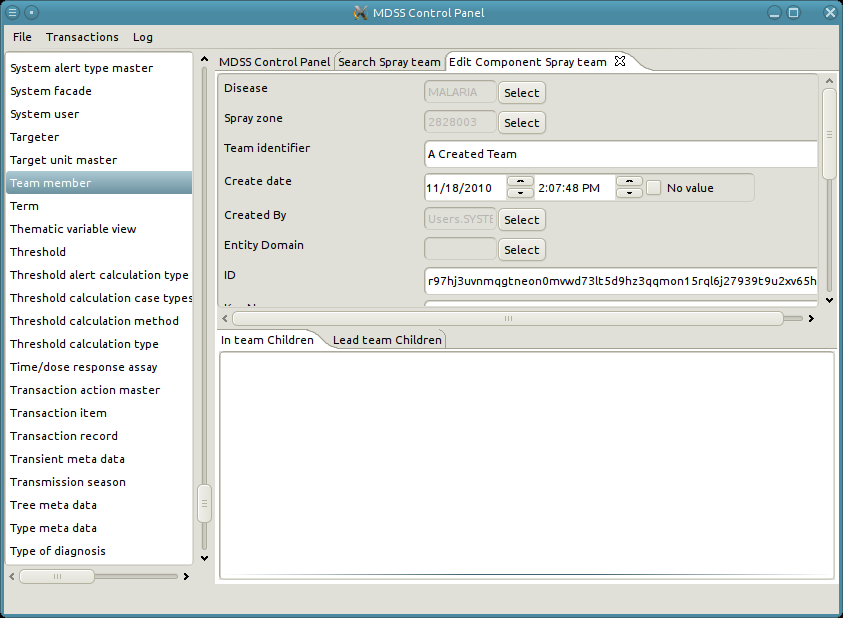
### Deleting relationships

Similar to deleting an object, deleting a relationship is done when viewing the relationship. See "Viewing a relationship" for details on viewing relationships.

1. Select the relationship object to edit.
2. A new tab with the details of the relationship opens.
3. Scroll down to the bottom of the tab and click "Delete."

# An example of deleting a relationship

This example deletes the relationship from the creation example.

1. First view the Spray team "A Created Team."
2. Right-click and select "Edit Relationship" on node "132."
3. Click "Delete."
4. View spray team "A Created Team" to ensure that the relationship was deleted.

Attributes

Runway supports many attribute types, with many different input mechanisms through the resolver.

# Booleans

Boolean attributes represent a simple true or false, selected in the UI by a radio button. The labels may not actually say “true” and “false;” instead they have specific labels customized for each attribute. Examples include:

* The Out of Stock attribute on Case Treatment Stock has labels “Yes” and “No”
* The Infected/Infectious attribute on Infection Assay Result has labels “Infected” and “Infectious”
* The Is Email Active attribute on Email has labels “Active” and “Inactive”

C:\helios\workspace\MDSSAdmin\doc\imgs\attributeBoolean.png

# Text

The flexibility of text attributes allows them to represent many different types of data. Names, ids, descriptions, notes, and others are all stored as text attributes. Each attribute has a maximum length which, if exceeded, will give an error message. To change the value, simply edit the text in the box. C:\helios\workspace\MDSSAdmin\doc\imgs\attributeChar.png

# Dates and Times

Storing dates and times requires consideration of two potentially difficult concepts: time zones and localization. On the web, time attributes display according to the time zone of the user. For example, if a user in GMT+1 stores a time of 07:30, and the server is located in GMT, it is stored as 06:30. If a second user in GMT-3 views the same attribute, he will see 03:30. With the synch resolver, however, time attributes are always displayed and stored relative to the time zone of the server.

Different countries and locales display dates in different ways. Is the correct format MM/DD/YYYY, DD/MM/YY, or something else? The resolver automatically detects the locale specified by windows and displays dates in the format for that locale. If the local format is unknown, check the task bar. Windows displays the current date in the bottom right corner, which can be used to ascertain the expected date format.

The Date/Time widget displays the date first, then the time. Values can be typed in or modified with the arrow buttons. However, the UI does not allow a date/time with a value to be cleared out. If clearing a date or time is necessary, check the “No value” box. Even though the widget still contains a value, if “No value” is checked, it will be cleared out on apply. C:\helios\workspace\MDSSAdmin\doc\imgs\attributeDate.png

# Numbers

Numeric values are basically divided into two categories: integers and decimals. Integer attributes, as the name implies, accept only whole numbers. The widget is a simple text box, but input is verified to prevent letters, symbols, or other illogical data. C:\helios\workspace\MDSSAdmin\doc\imgs\attributeNumber.png

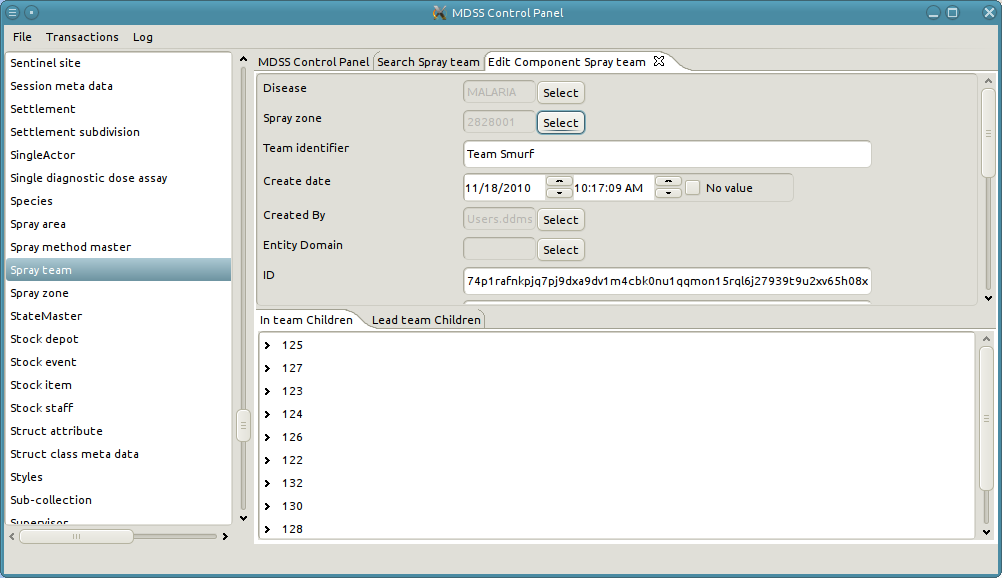
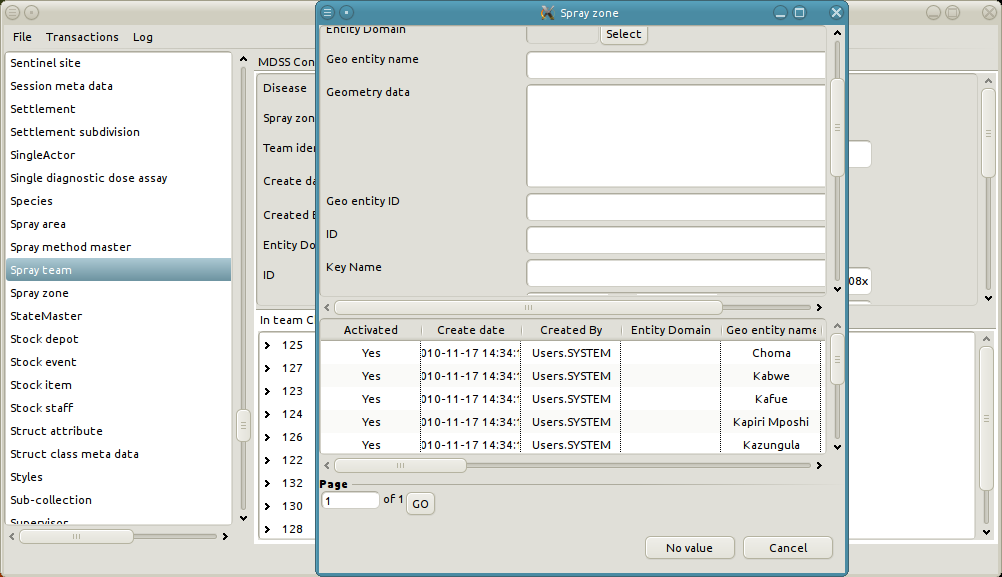
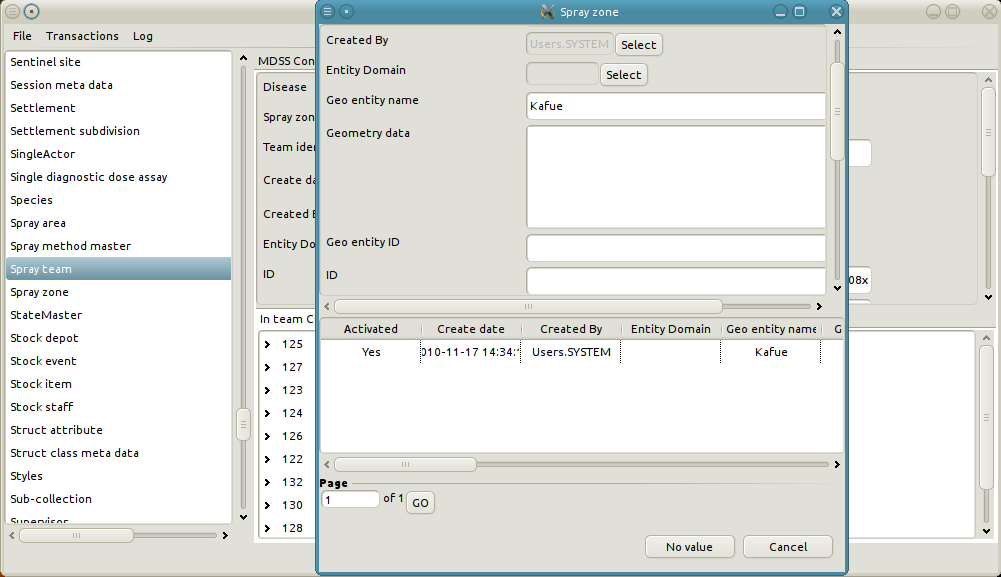
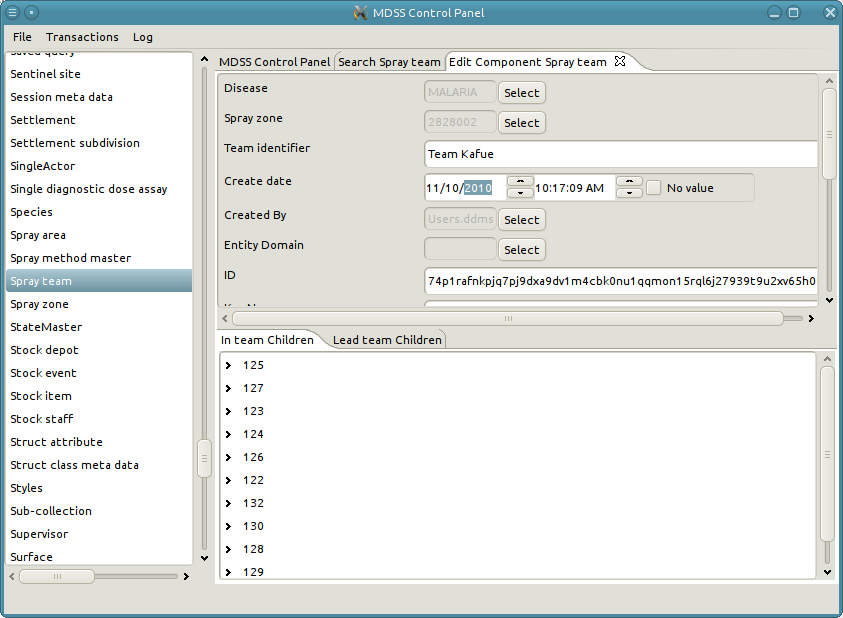
# Enumerations

An enumeration is used when acceptable values for an attribute are limited to a predetermined list. As an abstract example, the four seasons would be an enumeration consisting of Spring, Summer, Fall and Winter. In the resolver, enumerations are displayed as a drop-down list. Click the arrow to reveal the list and select the desired option.

C:\helios\workspace\MDSSAdmin\doc\imgs\attributeEnumeration.png

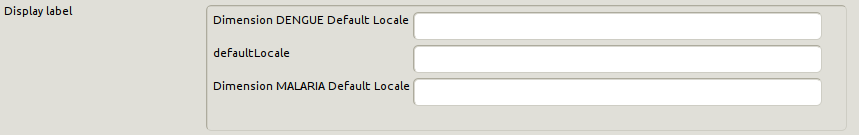
# References

Reference attributes enable objects to store the id of other objects for quick access in the future. Each reference attribute specifies the type of object that it expects. A recurring example of reference attributes in DDMS is geographic data, which is seen in assays, surveys, cases, and more. Ontology terms are another prevalent example, as they are also referenced by many different data types. Since reference attributes point to existing object instances, the input process involves several steps.

1. Start by clicking Select next to the reference attribute. 
2. This opens a dialog, which is used to search through all of the objects of the correct type. The top half of the dialog includes all of the attributes on the referenced data type, while the bottom half shows a table of search results.
3. Attributes can be input in the top half before to narrow the list of results. In the example, the geo entity name is set to Kafue. Click search to filter the list of objects to those that match the attribute specified in the form. In the example, only one Geo Entity has an entity name Kafue.
4. Once the desired instance is found, double-click the row to select it. Alternatively, the row can be selected by right-clicking and choosing select. The dialog will close and the reference attribute will be set to the new value.

To clear a reference attribute click Search to open the search dialog, but instead of searching for an instance, click the “No value” button at the bottom. The will close the dialog and clear the value on the form.

# Structs

A struct is an attribute that is semantically a single unit but is composed of several sub-attributes. As an example, a mailing address is a single attribute, but is composed of the sub-attributes Street, City, State, and Zip Code. In the synch resolver, struct sub-attributes appear as a group. Though the sub-attributes may be different types, their input widgets will behave like regular attributes. Fill them in as normal.

Transactions

A transaction is an atomic set of creates, updates, and deletes on one or more objects that collectively implement an operation in DDMS. A single transaction records contains one or more transaction items. For instance, the creation of a Spray team and assignment of its respective Spray operators requires multiple items, but is recorded as a single transaction. A transaction item is a single action which occurs during a transaction. The Spray team example contains multiple transaction items: creating the spray team itself, and creation of In Team relationships for each spray operator.

Transactions are important because the database will either perform all of the actions, or none of them. Before attempting a transaction, the database creates a save point. If any of the transaction items fail, the database rolls back to the save point. This all-or-nothing semantic ensures that data is not corrupted in the event of an error. See the Basic CRUD Operations introduction for more information on data integrity and corruption.

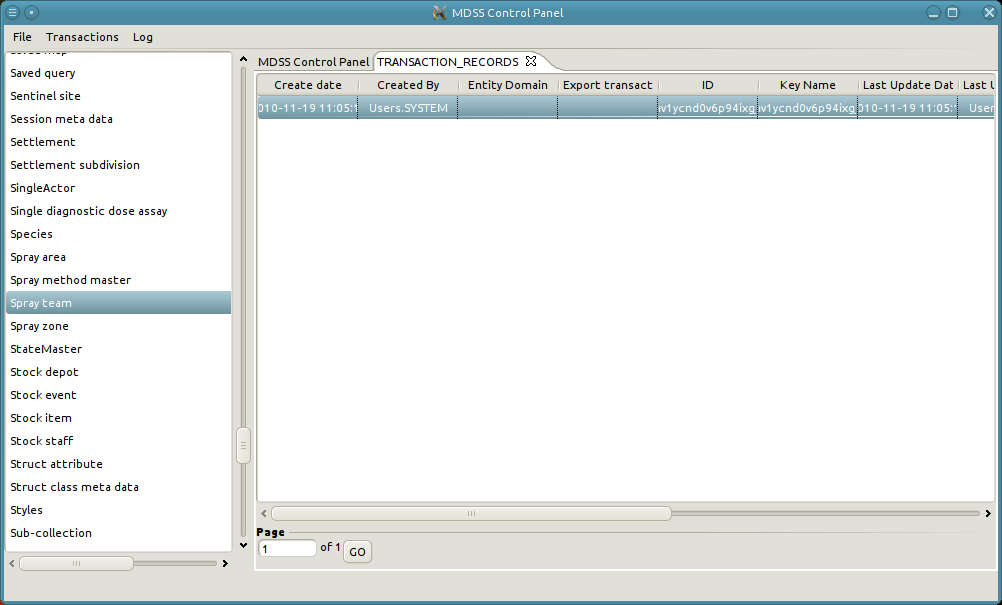
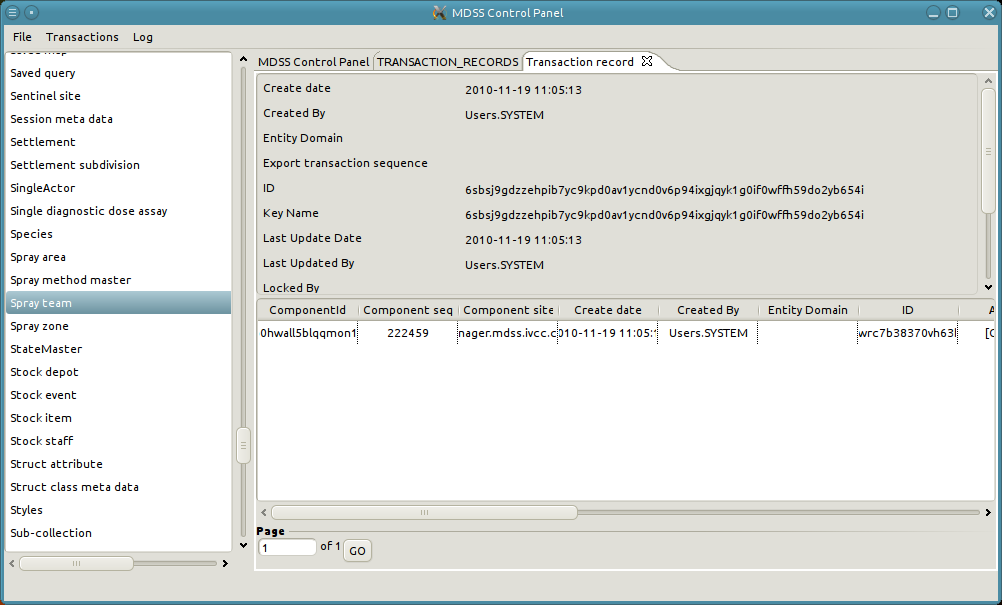
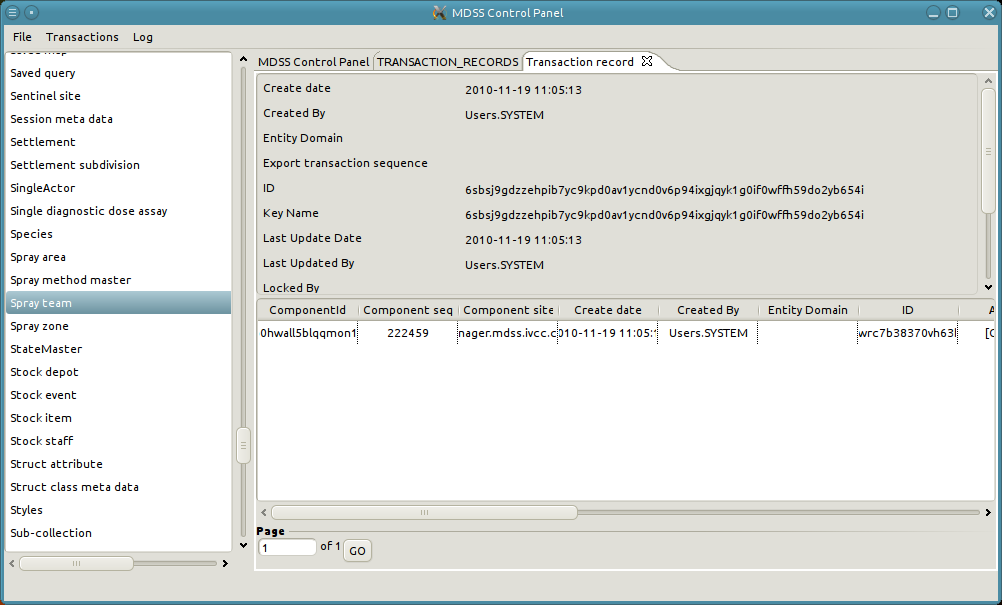
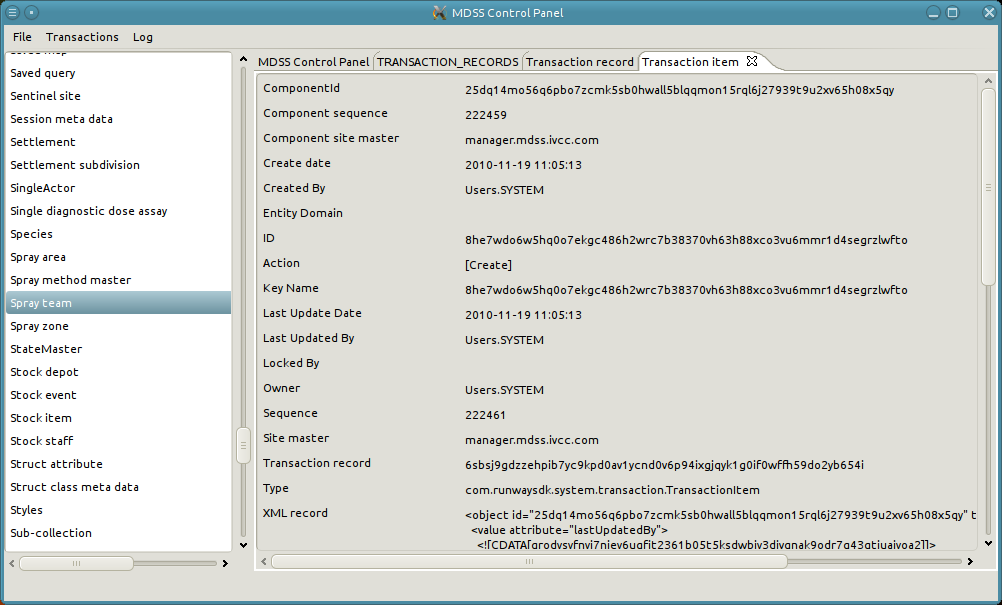
DDMS creates transaction records for every successful transaction. A transaction record contains all of the transaction items required to reproduce the transaction on another computer. Synchronization uses transaction records to share data between different installations of DDMS. The synchronization manager facilitates viewing, importing, and exporting transaction records.

# View Transaction Records

Viewing a specific transaction record reveals the specific information contained in the transaction.

1. Open the "Transaction" menu and click "View Transaction Records"
2. The main panel opens a paginated table listing all transactions in the system
3. Double click a row to open the details of the transaction record in a new tab
4. The bottom panel of the transaction record tab contains a list of transaction items in the record
5. Double click a transaction item to open its view in a new tab

# An example of viewing transaction items

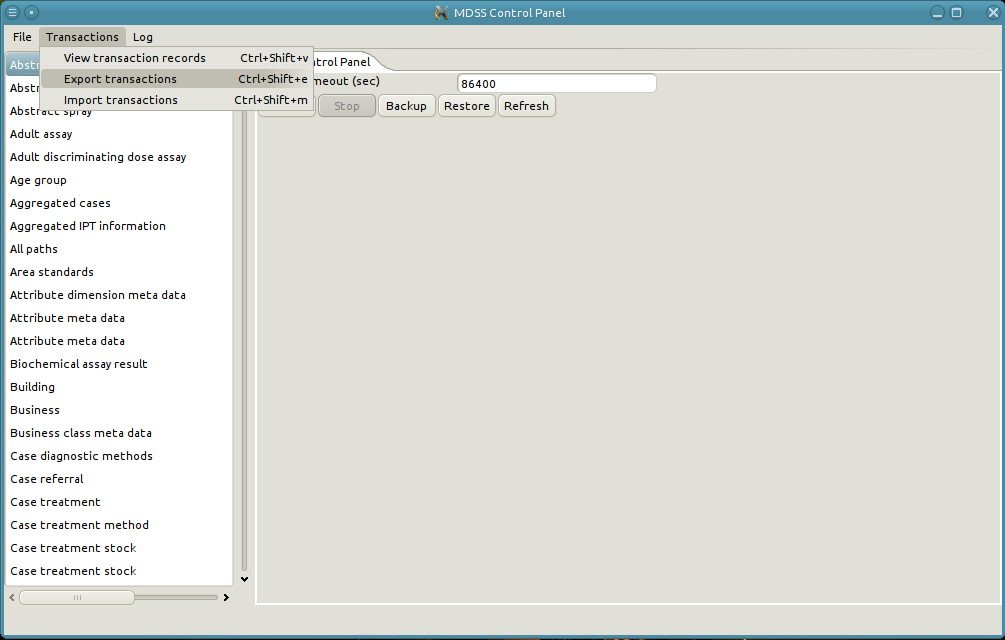
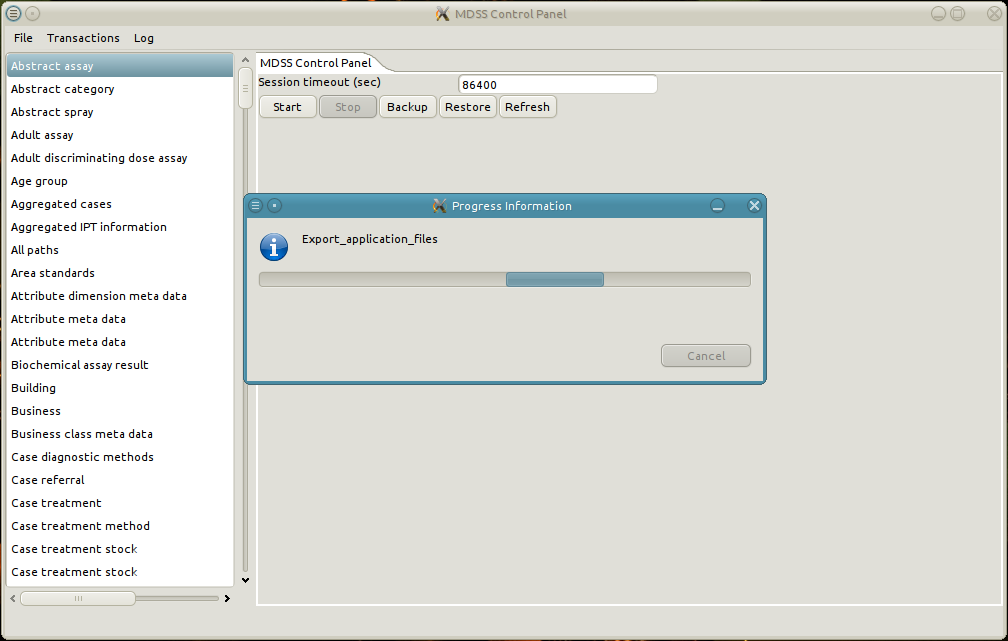
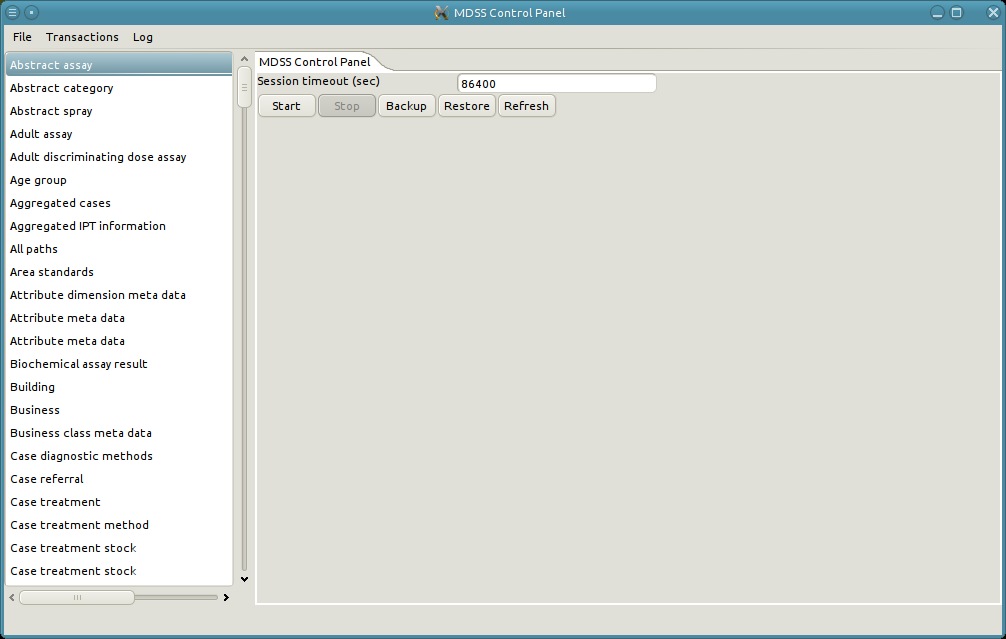
1. Open the "Transaction" menu and click "View Transaction Records"
2. A new tab opens with all of the transaction records in the system. In this example there is only one record.
3. Double click the row of the transaction record to view its details. Since this transaction record has not been exported, the “Export transaction sequence” field is blank. 
4. The table at the bottom contains all of the transaction items in a transaction record. In this example, the record has a single transaction item. Double click the transaction item to examine its details. 
5. A new tab opens with the details of the transaction item.

# Export Transaction

Exporting transaction facilitate data sharing between installations.

1. Ensure that the web server is shut down
2. Open the "Transaction" menu and click "Export Transaction"
3. A pop-up opens with three options: All, Range, and Not Exported
   * All: Exports all transactions from the node regardless whether or not they have already been exported. An importing node will skip all transactions which it has previously imported, so there is no danger of data duplication. However, exporting all transaction will create much larger export files.
   * Range: Exports transactions between the specified start and end sequence numbers. This option should only be used if an install requests the transaction of a specific export sequence number.
   * Not Exported: Exports all transactions which have not been previously exported. This is generally the preferred option.
4. Click “Choose File” and select a destination for the export file
5. Click “OK” to being the export
6. A pop-up appears with status information concerning the export. The pop-up closes when then the export completes.

# Example of exporting transactions

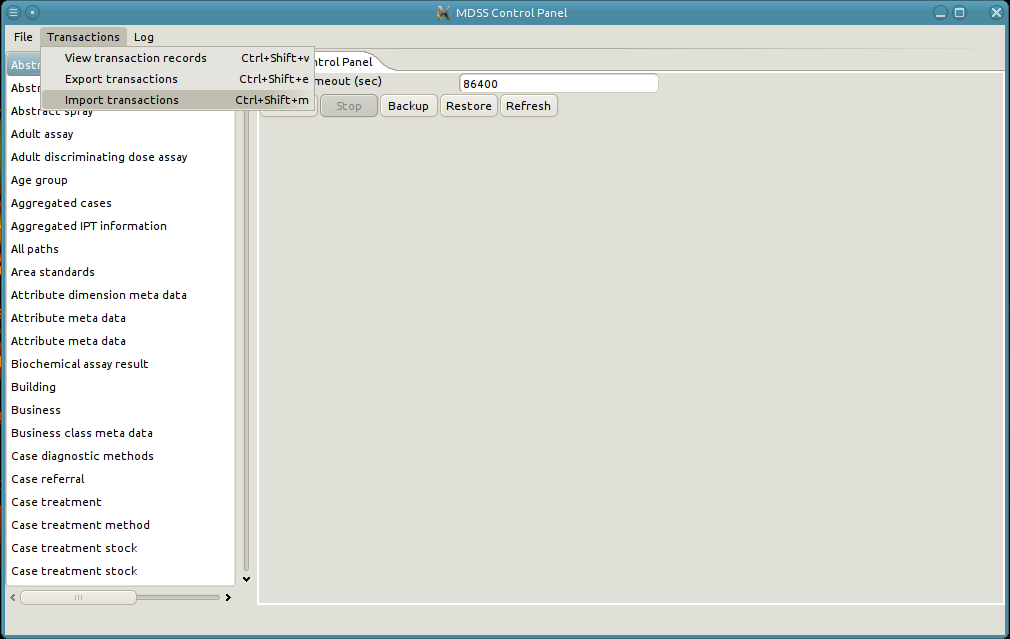
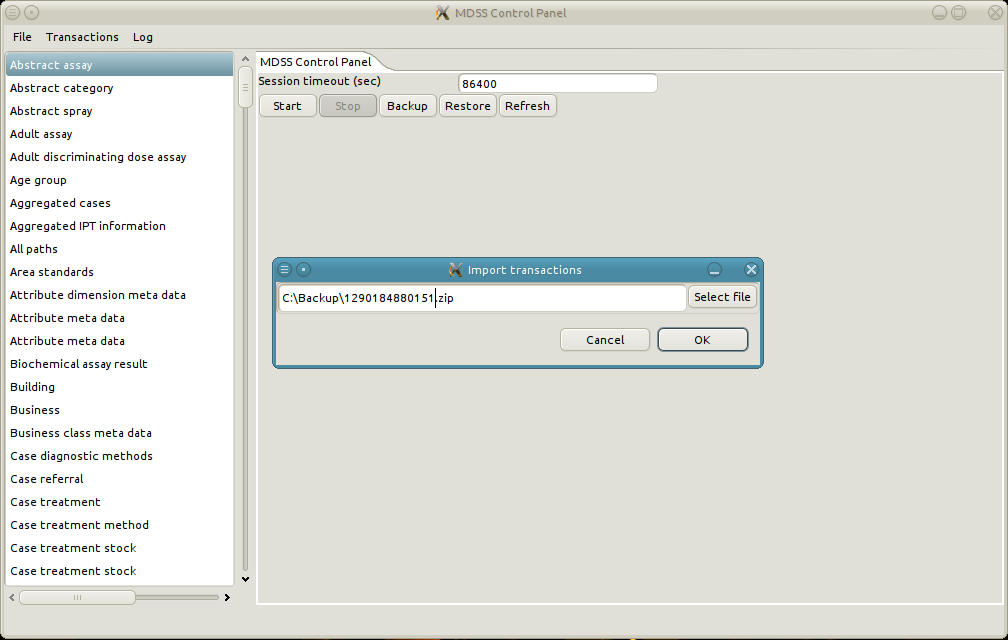
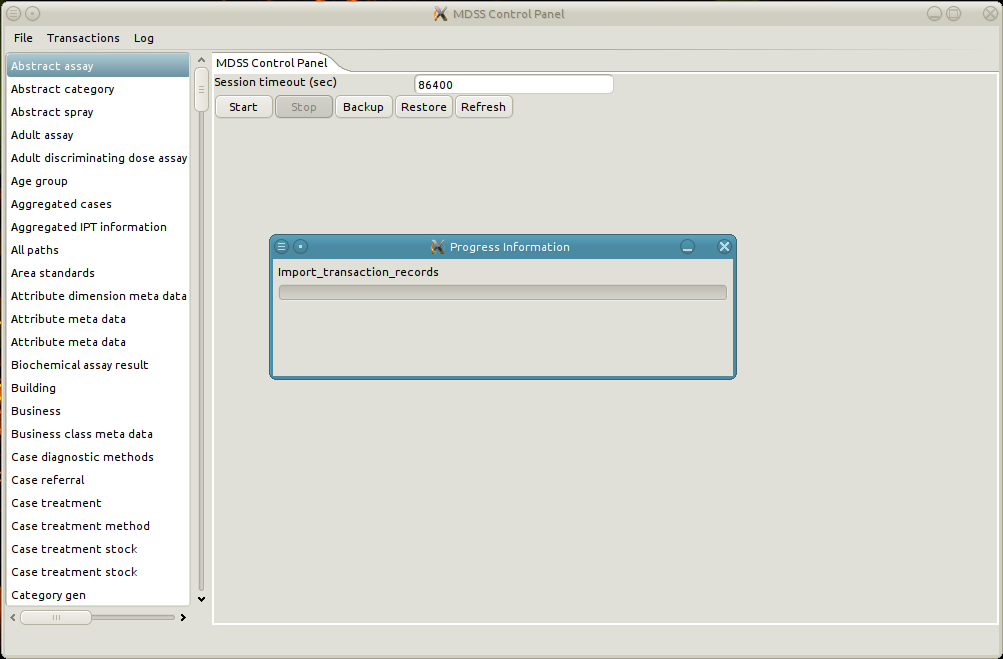
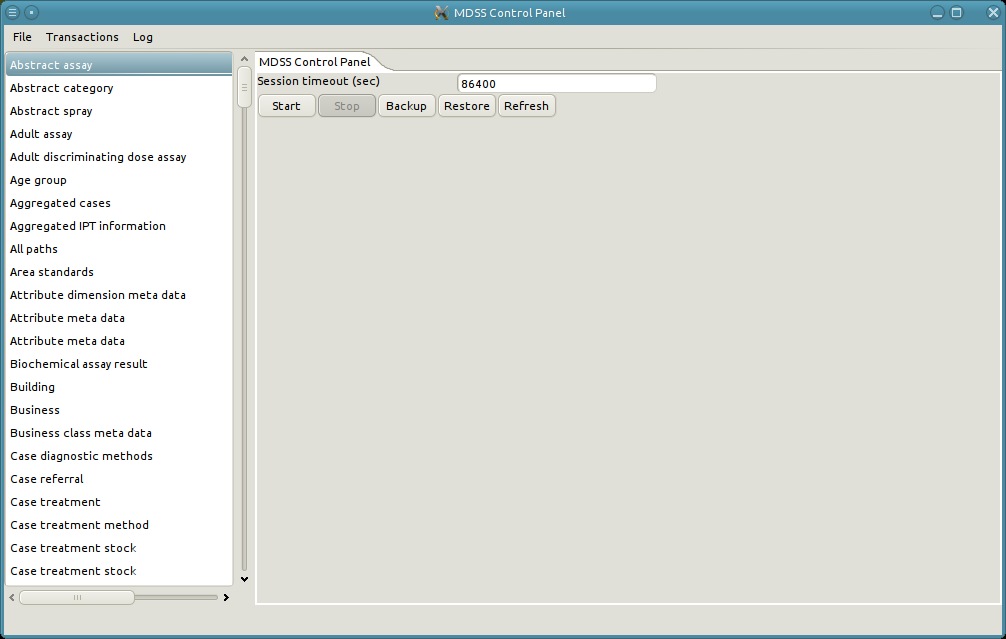
1. Open the "Transaction" menu and click "Export Transaction". Note that the server is shutdown.
2. Select the desired location of the export file and the sequences to export. Select “All” to export everything. Click "OK."
3. A pop-up appears with the status of the export.
4. The pop-up closes when the export is complete.

# Import Transaction

In order to receive data from a different installation, the transaction export file must be imported.

1. Ensure that the web server is shut down
2. Open the "Transaction" menu and click "Import Transaction"
3. A file selection dialog opens
4. Select the import file and click "Import"
5. A pop-up appears with status information concerning the import. The pop-up closes when the import completes.
6. Conflicts can arise during the import process. These must be resolved manually. See [Conflict Resolution] for a detailed approach for resolving conflicts.

# An example of importing transactions.

1. Open the "Transaction" menu and click "Import Transaction". Note that the server is shutdown.****
2. Select the file to import and click ok.
3. A pop-up appears with the status of the import.****
4. The pop-up automatically closes when the import is finished.****

Conflicts

When synchronizing data between installations, conflicts can arise between the two datasets. Perhaps both systems create users with the same username, or an imported IRS spray references a Spray Team that has been deleted. In these situations, the data is not consistent and must be modified to resolve the discrepancies. The primary purpose of the Synchronization Resolver is to provide the tools necessary to resolve any conflicts that may arise on import.

# Resolution Methods

The basic tools for resolution are the Edit, Create, and Delete procedures described earlier. However, each conflict represents a unique circumstance and must be considered carefully to ensure that the correct steps are takes for resolution. The next section discusses an example where two users are created with the same login, so one is renamed to resolve the conflict. Consider the scenario, however, where one user created the same account on two different installations. Even though the raised conflict is identical, in the second case the correct solution may be to delete one of the duplicated users, rather than leaving two users for the same person. This illustrates the need for a strong knowledge of DDMS as a whole when resolving conflicts. It is important to note that deleting objects through the resolver is especially risky, and this illustration is meant to demonstrate that one conflict can have multiple solutions depending on context. The illustration should not be viewed as an advocacy of delete operations.

# Conflict resolution control flow

When a conflict occurs during an import, the import process pauses and waits for the conflict to be resolved. A new tab opens showing the error message and the object that caused the conflict. The conflict must be resolved before the import process can continue. Note that while the resolver opens the conflicting object automatically, other object may need to be edited as well to preserve data integrity in the system. A single import may contain multiple conflicts. It is impossible to abort an import once the process has started.

## Uniqueness Constraints

While it is impossible to predict every potential conflict, some are more likely to occur than others. The previous example of two users with the same username demonstrates the most common conflict: a violation of a uniqueness constraint. Data is often checked in DDMS to ensure uniqueness. Examples include usernames, Spray Team IDs, Mosquito Collection IDs, and others. Suppose an installation creates a user with a username of john@mdss.com, but a second installation also creates a user with the same login. When either installation imports data from the other, a conflict will arise. In this case, the easiest solution is to change the username to something else, like john.doe@mdss.com.

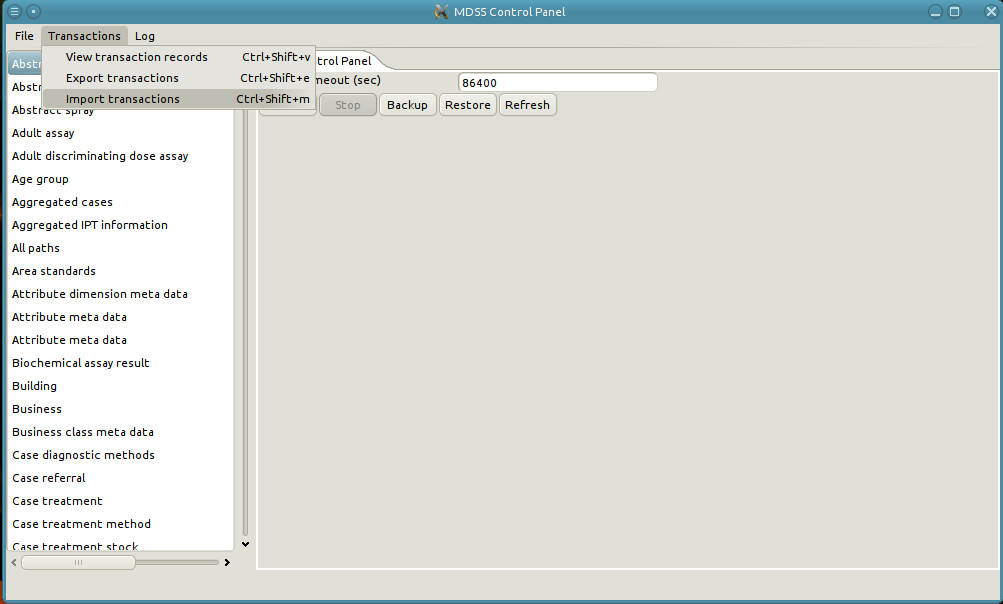
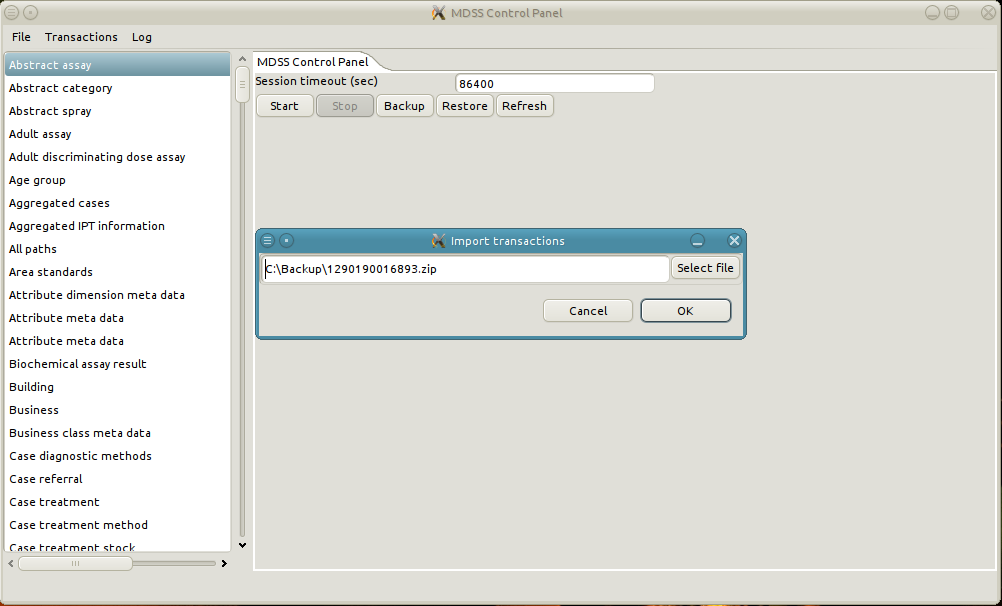
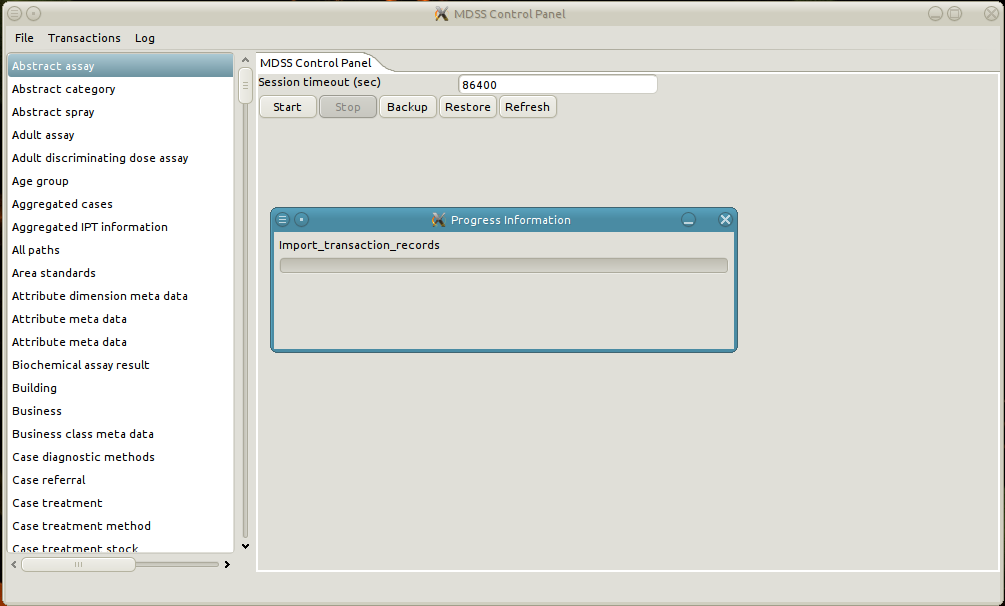
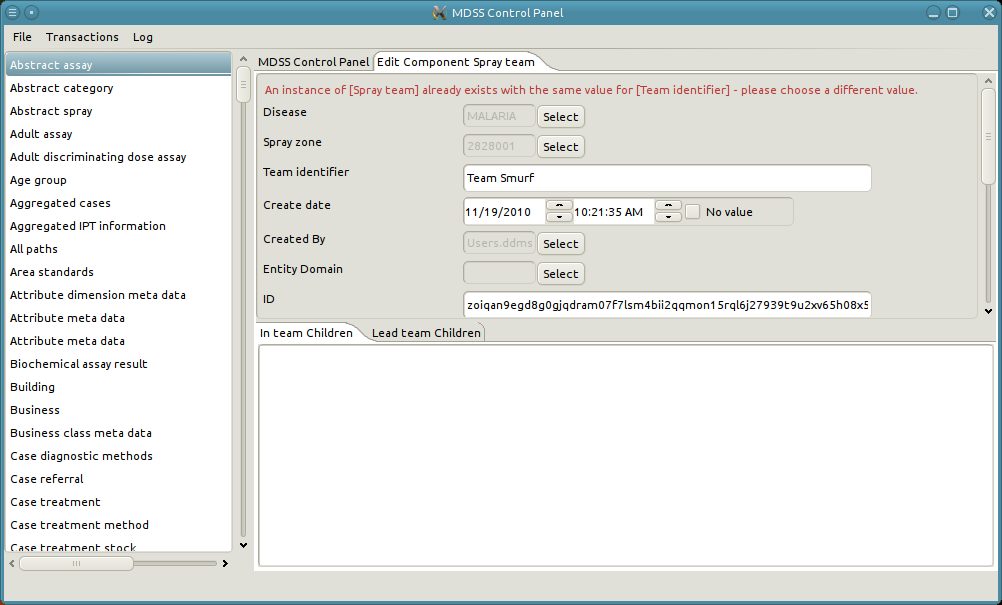
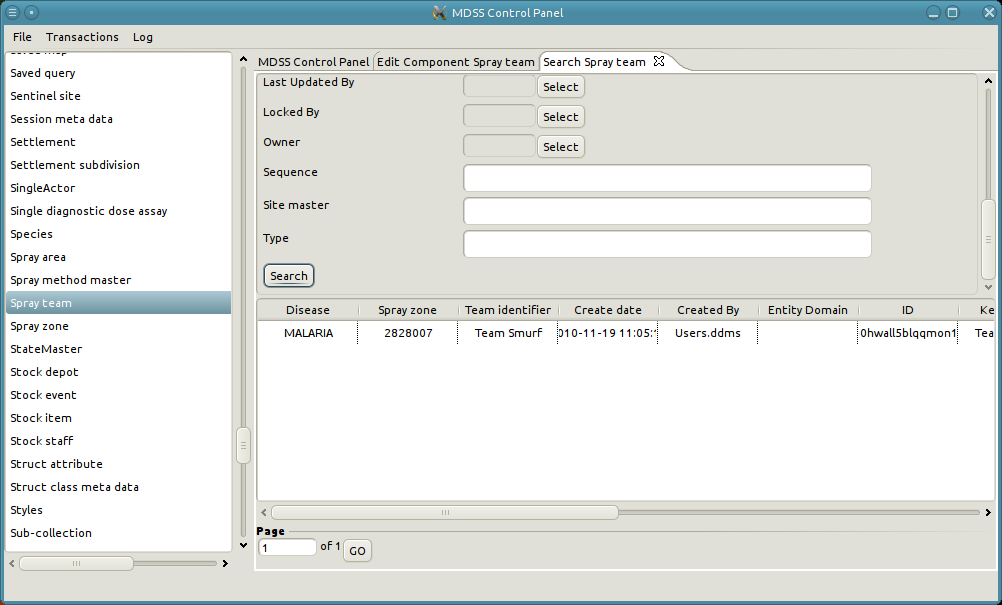
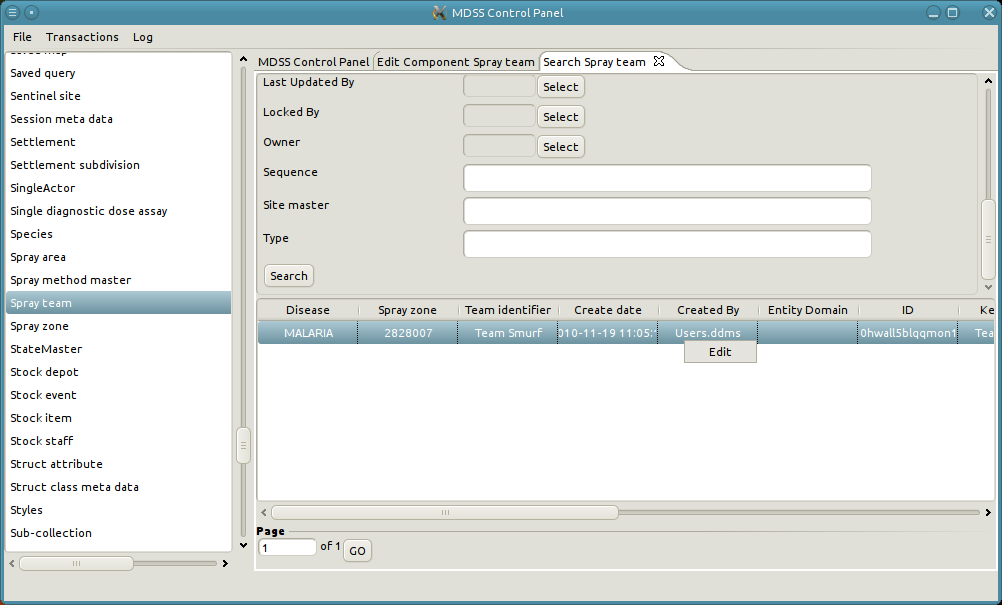
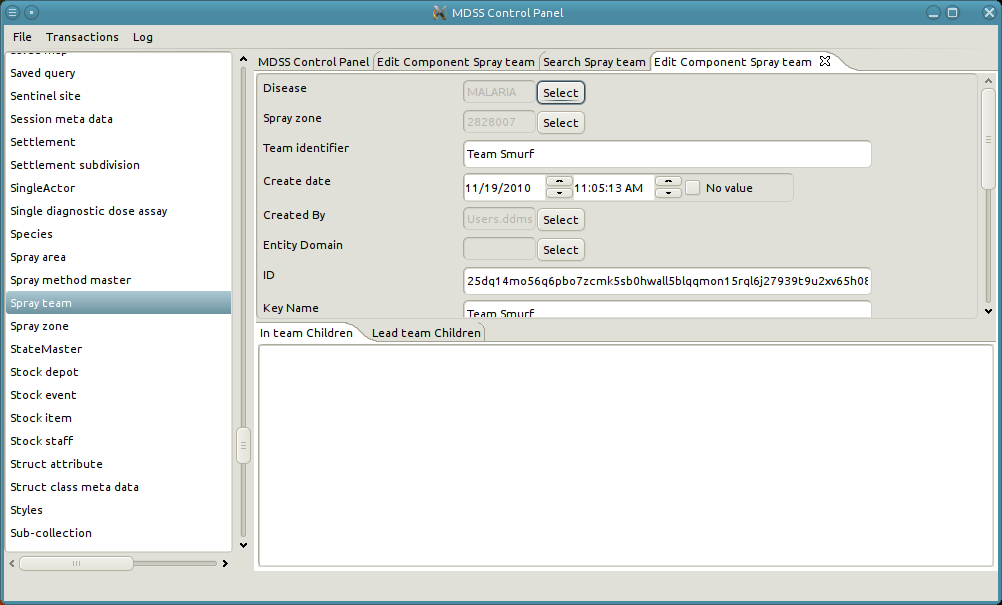
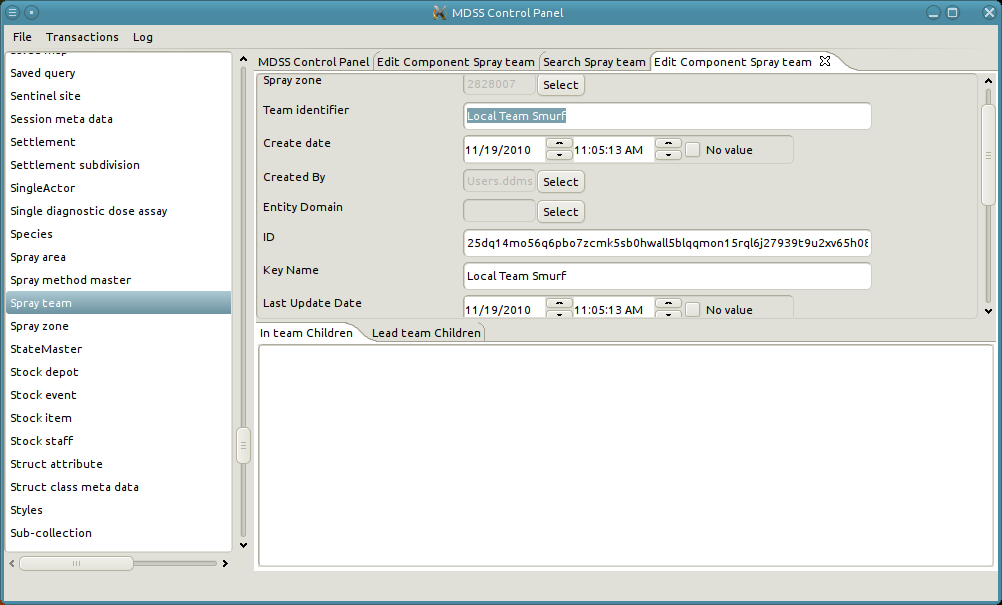
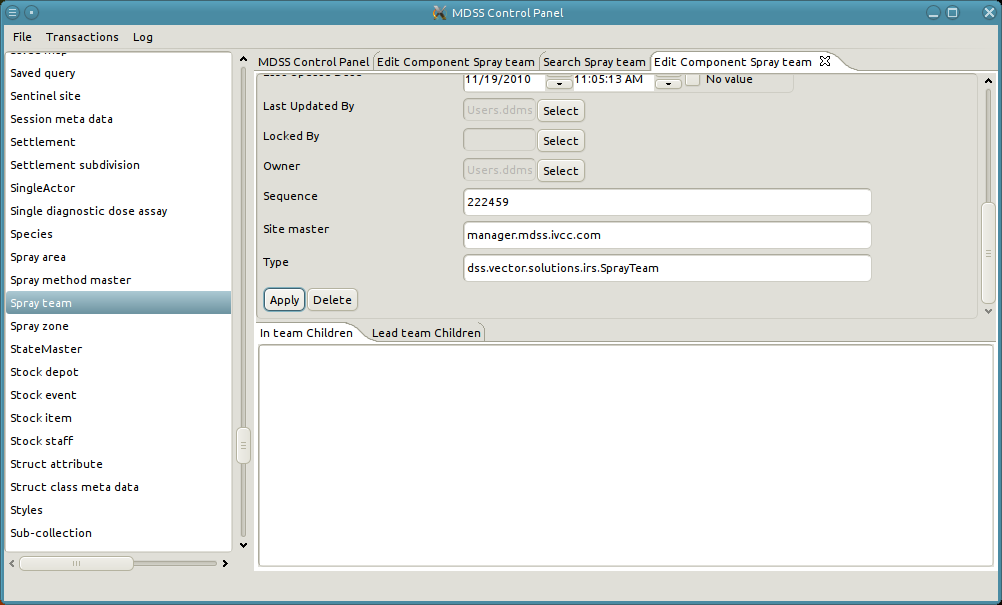
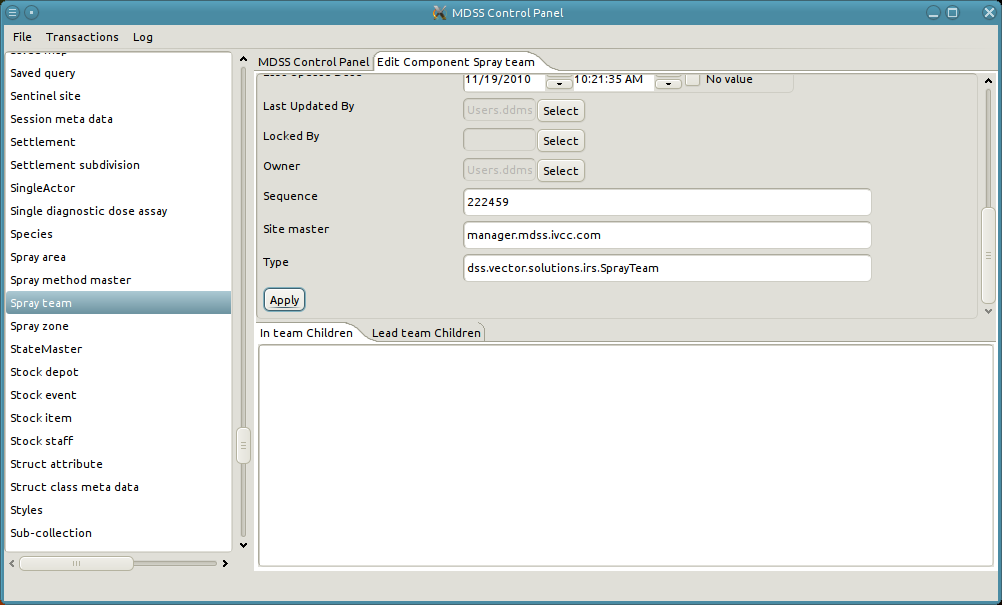
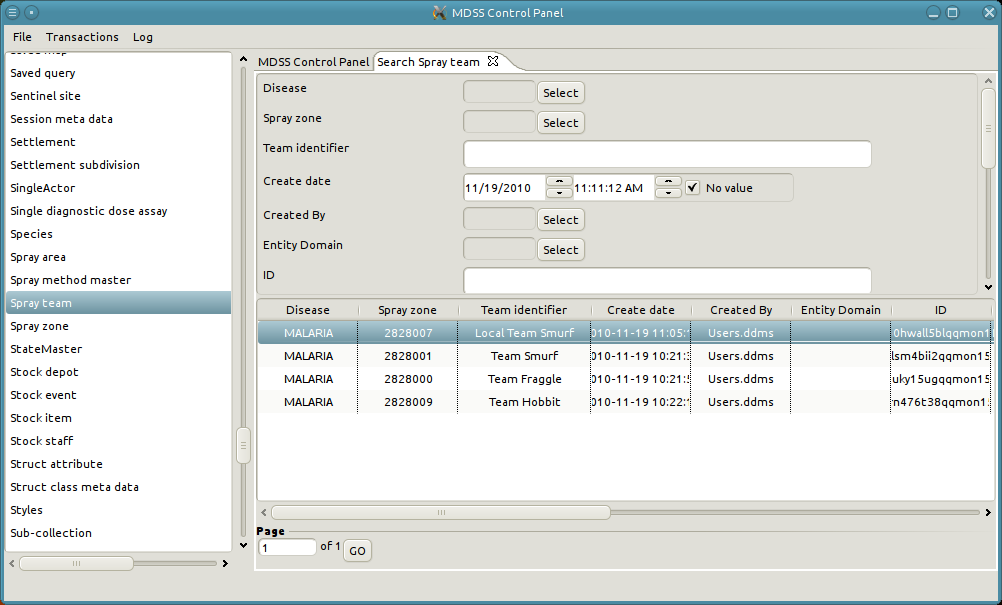
## Conflicts on relationships

In addition to all the conflicts which can occur on normal objects, relationships can have conflicts occur when the parent or child object is missing. These conflicts are especially difficult to resolve because the parent and child objects of a relationship cannot be modified. In general the best way to resolve such a conflict is to delete the imported relationship. However, if it is imperative that the relationship is persisted, then it possible to create a new relationship object with a different parent or child, and copy the values of values of the failed import relationship. Once the values have been copied, apply the new relationship and delete the original conflicted relationship.

## Resolving a Conflict

1. The import pauses on conflicts
2. A new tab opens viewing the conflicting object, with the error message at the top of the tab.
3. Resolve the conflict with the techniques described in the “Basic CRUD Operations” section. The specific steps necessary for resolution differ on a case-by-case basis, and may involve modifying several objects or relationships.
4. Assuming that the conflict has been resolved, clicking "Apply" on the conflicting object will automatically resume the import process. If the conflict has not been resolved then an error message will pop-up and the import will continue to wait until resolution.

## An example of resolving a conflict

1. Open the "Transaction" menu and click "Import Transaction"
2. Select the import file and click "OK."
3. A pop-up opens with the status of the import.
4. On conflict, the status pop-up closes and a new tab opens describing the conflict. In the example the conflict arises from a naming collision on the team identifier of the Spray team. Specifically, the import contains a Spray team called "Team Smurf", but there is already a Spray team of the same name.
5. Change the name of the local "Team Smurf" to a new name. Start by searching existing spray teams.
6. Edit the local "Team Smurf".
7. A new tab opens with the details of the local spray team. 
8. Change the Team identifier and the Key Name to "Local Team Smurf"
9. Click “Apply.” 
10. Finally, click Apply on the conflicting object. Since “Team Smurf” is available again, the import will continue. The object closes and the status pop-up returns. 
11. When the import is finished the status pop-up automatically closes. Search Spray teams to see both the newly imported team and the renamed team.

Logging

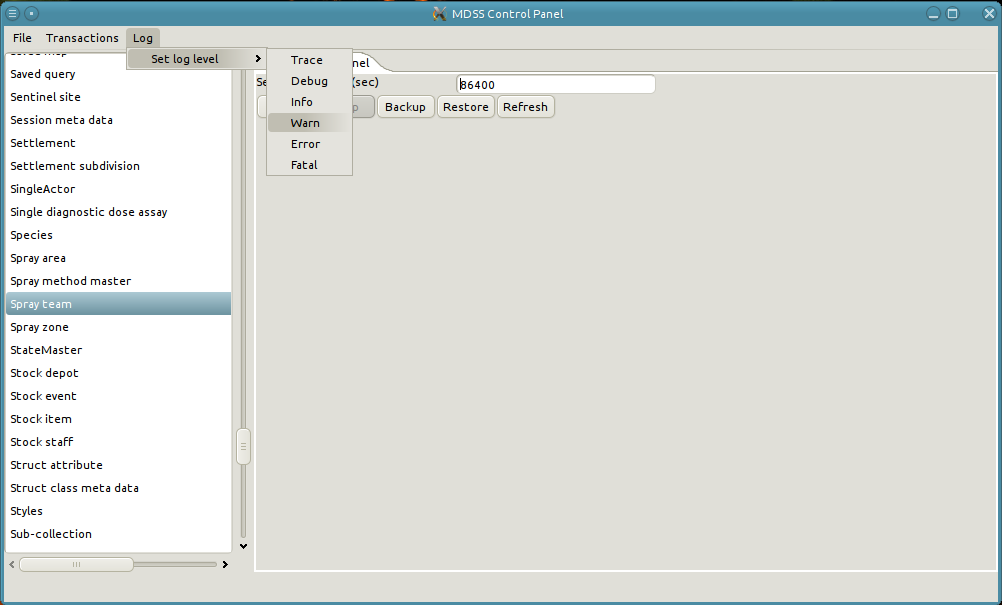
DDMS features a logging system that records data about application usage and saves the data to files on the computer called logs. The log files include data ranging from normal usage to serious errors and are recorded as one log file per day. Technical support can then use the log files to troubleshoot problems that have occurred within DDMS.

There are six different log levels, each one representing a different level of verbosity and seriousness. At the lowest level is Trace, which records virtually all usage but has the tendency to make the system slow and consume large amounts of disk space. At the highest level is Fatal, which records only the most serious errors but provides the least amount of contextual information. The levels between Trace and Fatal, in order of least to most serious, are Debug, Info, Warn, and Error. The right level can be chosen to provide the most meaningful compromise between verbosity and seriousness for a given problem. By default the log level is Error.

# Changing Logging Level

When a problem occurs in DDMS that requires technical support, recreate the problem under a more verbose log level provides extra contextual information, which can be very useful for debugging. To change the log level:

1. Open the “Log” menu item
2. Mouse over “Set Loge Level”
3. Click on the desired log level
4. Restart the server
5. Recreate the error
6. Repeat the process to reset the log level back to Error.



Logs are located in the directory C:/MDSS/logs/ with the current log file as log.xml and older log files named relative to their date. A user may select the current log file, representing today's DDMS usage, or any other number of log files and send them to technical support\* for analysis.

\* This process has not been completely defined.