GIS Core Database Revision:

# Update Data Resource Site Application Administrator's Guide

February 14th, 2002

Version 2.0

Robert Maki
GIS Database and Infrastructure Supervisor
Minnesota DNR
Management Information Services Bureau

## 1.0 Introduction

This document describes an integrated system level application operating at the Minnesota Department of Natural Resources, which maintains the currentness of fileserver based GIS data resources within the organization. It is one subsystem of the Revised GIS Core Architecture initiative. This document is written for a very technical audience, specifically persons who will have to troubleshoot, extend, or replace the application being described here. Any reader of the document must have an understanding of the overall Revised Core Architecture to make any sense of it and are encouraged to review: "Revised Core Database General Architecture Design," and "Minnesota DNR Data Resource Site Specifications."

## 2.0 General Overview

The Update Data Resource Site application (update_drs) is a suite of programs which update the data resources and associated descriptive metadata of a Minnesota DNR Data Resource Site (DRS). A DRS is a "data center" where data are explicitly stored or otherwise referenced (i.e. centralized database server sources). DRS site administrators are empowered to describe the desired conditions of a data resource site with regard to data content. The application obtains the DRS "desired conditions" specified by the site administrator, compares them to the actual conditions present on the site, and acts to adjust the site content to reflect the desired conditions. A data flow diagram (DFD) of the process is provided in Figure 1.

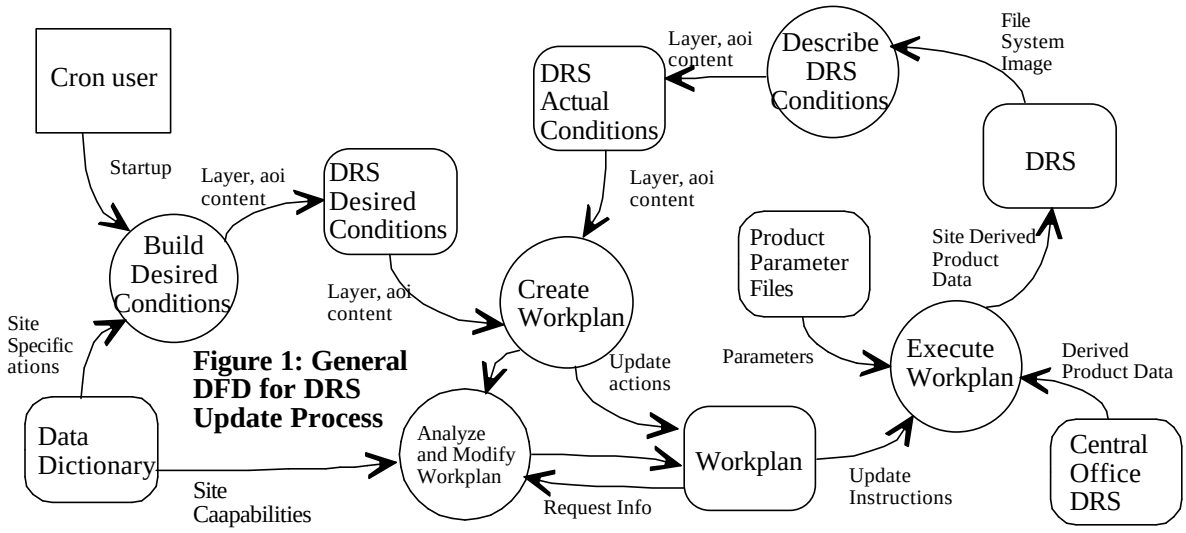


**Figure 1: General DFD for DRS Update Process**

The system draws data from the Central Office DRS (referred to as "Drsmain"), retaining format and content but varying in spatial extent (area of interest, or "AOI"). Each DRS-Layer

combination has its own AOI.

The application operates on a Solaris operating system and includes components written in Perl and Arc Macro Language (AML). It interacts with three specific data storage environments: 1) an Oracle database which stores and presents information on DRS desired conditions (the DNR Data Dictionary, or DD), 2) a set of ASCII formatted text files that describe the states of the desired and actual sites, and instruction sets for update and 3) a variety of filesystem-based GIS data structures that hold the data being distributed/updated/removed.

The application operates in a fixed linear sequence of steps to execute the update process. The main driver for the application is called from a scheduled UNIX cron job. Update schedules are set at the operating system level. Each site is considered in turn, and subject to the site update process. The process itself features a fairly extensive set of error condition tests which may be either fatal or non-fatal in nature. Error conditions are written to a daily error report with subsections headed with a DRS name and date stamp. Similarly, specific update activities are written to a separate log.

## 3.0 Major Process Descriptions

The application consists of twenty-two separate program modules. Thirteen of these actually perform the work, four are drivers for the others, and five others perform common tasks within the environment. Figure 2 lists the programs and the hierarchical order in which they are called. Working (non-driver) modules are listed with an "*".

**Figure 2. Update DRS Program Structure Chart**

```
queue_update_job.pl                                  Common modules:
      drive_drs_update.pl
            drive_desired_cond.pl                    common.pl
                  make_desired_layer_def.pl*         error_rep.pl
                  Make_desired_store_def.pl*         mk_early_term.pl
                  make_desired_aoi.pl*               rm_early_term.pl
                  Make_desired_lay_stat.aml*         common_var.aml
            create_drs_workplan.pl*
            triage_update_proc.pl*
            update_site.aml*
            update_drs_def.pl*
            existing_cond_driver.pl
                  gen_layer_status.pl*
                  update_drs_alyer_list.pl*
                  update_drs_aoi.pl*
                  Update_drs_lib.aml*
                  Update_image_cat.aml*
```

## 3.1 Metafile Processing

Any discussion of major processes must be prefaced by a description of the various types of metafiles being processed. The application processes sets of metadata describing the desired and actual status (conditions) of a data resource site. Desired conditions originate in the DNR Data Dictionary, an Oracle-based data registry environment. These conditions are translated into two metafiles: 1) a data_layer_def file, and 2) an AOI file. A third metafile class, the layer_status file is generated based on a combination of AOI and existing conditions on the Central Office DRS. Metafiles constituting the actions to be taken in a site update process round out the list of system metadata types.

### 3.1.1 data_layer_def file

The data_layer_def file format is formally described in "Minnesota DNR Data Resource Site Specifications." This simple comma delimited ASCII file adheres to the following format:

```
[layer_desc],[DNR_Unit],[tiling_scheme],[product_type],[resource_description],
[vendor_object_type],[server],[instance],[product ID],[thematic class id]
```

data_layer_def records example:

```
Minnesota County Borders,gen,state,2,bdry_counpy3,
      polygon,null,null,220000020201,22

1:24000 Quadrangle Index,gen,state,2,indx_q024kpy4,
      polygon,null,null, 210000020201,21
```

### 3.1.2 aoi files

AOI files are ASCII files which contain a list of lower case four character county abbreviations.[1] They are named in the following manner:

```
[resource description]-aoi.txt
```

### 3.1.3 layer status files

Layer status files are comma-delimited ASCII files used to express tile-based status for each layer. Each layer-site has a layer status file which indicates tile id, date, and size in Kb. They adhere to the following format:

[tile id], [date], [size in Kb]

---

[1]Area of Interest is expressed in whole county increments

They are named in the following manner:

```
[resource description]-layer.txt
```

These three metadata classes exist in two types of locations: 1) desired conditions, and 2) data resource sites. Their physical placement on the filesystem is described in Figures 3a and 3b.

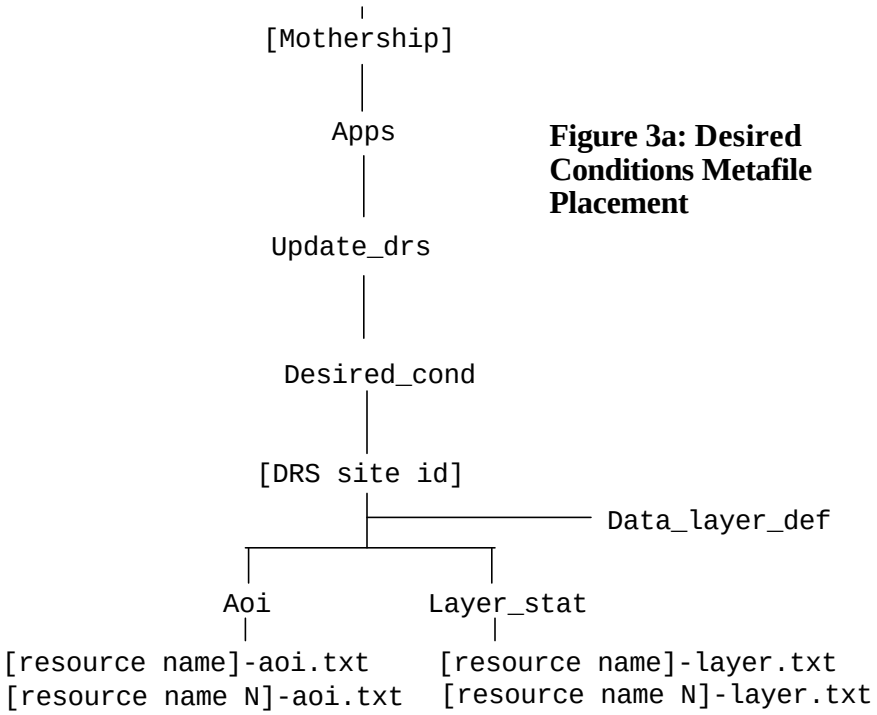


**Figure 3a: Desired Conditions Metafile Placement**

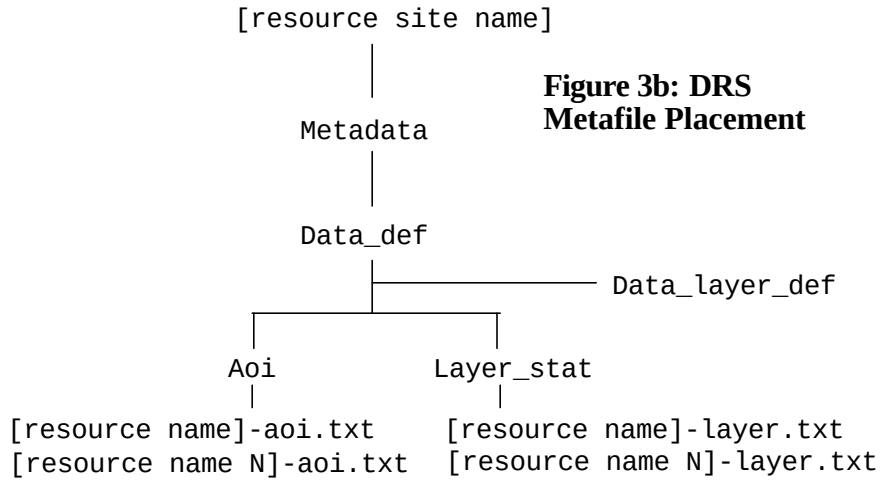


**Figure 3b: DRS Metafile Placement**

These three metadata classes define everything that the application needs to know about a site: layer content (existence, absence, format, tiling scheme, administrator), area of interest (by county), and layer status by tile (including date information). A set of these metafiles is developed for the desired conditions and compared to a set which reflects actual conditions. Differences between the two are expressed in a "workplan" document which contains the instruction sets for performing the actual site updates.

3.1.4 Workplans

Workplans are XML style documents which adhere to the format described in Figure 4. Workplans have three primary subsections: 1) addlayer, 2) droplayer, and 3) modifylayer. The first two are essentially lists of tiles to be added or dropped. The third is somewhat more complex in that modifying layers may entail copying tiles, deleting tiles, or making new tiles (clips of statewide data).

**Figure 4: DRS Update Workplan Document Type**

```
<workplan>
<addlayer>
<resource_name>[resource name]</resource_name>
<extent>[state | subset]</extent>
<tile>[tile id]</tile>
<tile>[tile N]</tile>
</addlayer>
<droplayer>
<resource_name>[resource name]</resource_name>
<tile>[tile id]</tile>
```

```
<tile>[tile N]</tile>
</droplayer>
<modlayer>
<resource_name>[resource name]</resource_name>
<extent>[state | subset]</extent>
<copytile>
<tile>[tile id]</tile>
<tile>[tile N]</tile>
</copytile>
<deltile>
<tile>[tile id]</tile>
<tile>[tile N]</tile>
</deltile>
<maketile>[tile id]</maketile>
</modlayer>
</workplan>
```

Workplan actions are read by a sub-process which executes them. Workplans can have any number of addlayer, droplayer, and modlayer sections.

Workplans are located at $MOTHERSHIP/applications/update_drs/workplans

and are named as:

`[DRS ID]_workplan.txt`

3.1.5 Action Layer Lists

Update processes are also informed by a list of layers that constitute the set of data layer objects that are being acted upon. Action layer lists are stored at:

$MOTHERSHIP/applications/update_drs/desired_cond/<drs ID>/`act_layers`

Action layer lists are ASCII text files which store a set of resource names. The following is an example:

```
lake_usgspy2
bdry_counpy2
```
etc.

Action layer lists are system generated by the program “triage_update_proc.pl” or, in the case of a Drsmain site update, by “drive_main_partial.pl.” They are included in the discussion here because they can be used to selectively guide a manual update process, especially when rebuilding the main DRS and recovering from a disrupted update process (Section 5).

3.2 Individual Process Descriptions

### 3.2.1 queue_update_job.pl [NOT IMPLEMENTED AT THIS TIME]

Description: This master driver program (in Perl) is executed from a scheduled cron job in UNIX. It queries the data dictionary and returns a list of data resource sites that should be updated along with associated system locations.

Inputs: data dictionary query results
Outputs: None
Calls made: drive_drs_update.pl

### 3.2.2 drive_drs_update.pl

Description: This is the master driver operating per data resource site.

Inputs: arguments(data resource site name), drs location from the data dictionary
Outputs: None
Calls made: drive_desired_cond.pl ,create_drs_workplan.pl, update_site.aml, update_drs_def.pl, existing_cond_driver.pl

### 3.2.3 drive_desired_cond.pl

Description: Calls the programs which build desired conditions.

Inputs: arguments(data resource site name)
Outputs: None
Calls made: make_desired_layer_def.pl, make_desired_aoi.pl, make_desired_lay_stat.aml

### 3.2.4 make_desired_layer_def.pl

Description: creates a data_layer_def for a specific layer from the desired conditions present in the data dictionary. It queries the Data Dictionary to create the layer list.

Inputs: arguments(data resource site name), Data Dictionary desired conditions
Outputs: a desired conditions data_layer_def file.
Calls made: None

### 3.2.5 make_desired_store_def.pl

Description: creates a file with relevant connection information for non-file server based GIS sources. It queries the database_src file in the Data Dictionary.

Inputs: Data Dictionary
Outputs: a data_store_def file in the desired conditions location
Calls made: None

### 3.2.6 make_desired_aoi.pl

Description: creates AOI files for each layer in the desired conditions data_layer_def file. It queries the Data Dictionary to develop the AOI's.

Inputs: desired conditions data_layer_def file, arguments(data resource site name)
Outputs: a suite of AOI files
Calls made: None

### 3.2.7 make_desired_lay_stat.aml

Description: Creates layer status files based on site-layer AOI. Builds desired layer status files from the layer status file records that exist on the Central Office DRS.

Inputs: arguments(data resource site name), desired conditions data_layer_def file, aoi files, Central Office DRS layer_stat files, tile x-ref tables (at $MOTHERSHIP/apps/gen_app_support/search_indexes/info!arc![tile scheme name]_xref
Outputs: a suite of layer_stat files
Calls made: None

### 3.2.8 create_drs_workplan.pl

Description: creates a workplan document for a site by comparing the desired and actual conditions of a site. Changes to the three principle metafile types are considered [layer presence/absence, change in layer status (tile date, tile prsence/absence), and area of interest change]. Additionally, layer inclusion in a workplan can be triggered by: change of tiling scheme, or change in data format for a layer.

Inputs: desired conditions layer status, desired conditions AOI, desired conditions layer list, actual conditions layer status, actual conditions AOI, actual conditions layer list, arguments(data resource site name, data resource site location)
Outputs: a workplan document by DRS
Calls made: None

### 3.2.9 triage_update_proc.pl

Description: reads a workplan document and determines if the actions stored within it will compromise the DRS being updated. The following tests are made: 1) too much data being deleted, 2) too much data requested, and 3) not enough disk space on target location. Case 1 will result in the entire process aborting. At the time of this writing, an excessive delete request is considered to be an amount that would take more than 10 hours to replace under the current linespeed rating. Case 2 will result in a rescaling of the addition request to an amount that will be transferrable within 10 hours under the current linespeed request. Layer requests are sorted by total data volume and iteratively pared down tile by tile until the request is considered to be a manageable size (beginning with the largest layer). Case 3 operates the same way, except that it

uses disk space criteria rather than add request-linespeed. In this case, the application will always preserve 100 Mb of disk space on the target device.

Inputs: a DRS workplan, $mothership layer status files, linespeed rating obtained from the data dictionary.
Outputs: an error report, if necessary
Calls made: none

3.2.10 update_site.aml

Description: Reads a workplan, and updates a data resource site

Inputs: a DRS workplan document, arguments(data resource site name, data resource site location)
Outputs: An updated DRS site
Calls made: None

3.2.11 update_drs_def.pl

Description: copies a new data_layer_def file to the DRS from the desired conditions location

Inputs: arguments(data resource site name, data resource site location)
Outputs: a revised data_layer_def file on the DRS
Calls made: None

3.2.12 update_dsk_apps.pl [RETIRED]

3.2.13 existing_cond_driver.pl

Description: Main driver for establishing existing conditions. This is necessary to ensure compliance between the DRS and its associated metadata

Inputs: arguments(data resource site name, data resource site location)
Outputs: None
Calls made: gen_layer_status.pl, update_drs_layer_list.pl, update_drs_aoi.pl, update_drs_lib.aml

3.2.14 gen_layer_status.pl

Description: Creates layer status files based on actual filesystem conditions. Invoking the active layer override option causes a full layer status file rebuild for the site.

Inputs: DRS data_layer_def file, arguments(data resource site name, data resource site location, active layer override [Y])
Outputs: A suite of layer_status files for a DRS
Calls made: None

3.2.15 update_drs_layer_list.pl

Description: Eliminates records from a DRS data_layer_def file based on existence of the data at the file system location.

Inputs: DRS data_layer_def file, arguments(data resource site name, data resource site location)
Outputs: A revised DRS data_layer_def file
Calls made: None

3.2.16 update_drs_aoi.pl

Description: Transfers a new set of AOI file from the desired conditions location

Inputs: Desired conditions AOI files, arguments(data resource site name, data resource site location)
Outputs: a new suite of DRS AOI files
Calls made: None

3.2.17 update_drs_lib.aml

Description: Reconfigures the DRS librarian environment to reflect actual conditions on the site. Invoking the active layer override option causes a full librarian environment rebuild for the site.

Inputs: DRS data_layer_def file, DRS layer_status files, $ARCHOME/tables/libraries file, arguments(data resource site name, data resource site location, layer override option [Y])
Outputs: A revised $ARCHOME/tables/libraries file, new or eliminated library “database” directories
Calls made: None

3.2.18 update_image_cats.aml

Description: Reconfigures the image catalog environment to reflect actual conditions on the site. Reads the data_layer_def file and builds new catalogs for all image themes. Invoking the active layer override option causes a full image catalog rebuild for the site.

Inputs: DRS data_layer_def file, DRS layer_status files
arguments(data resource site name, data resource site location, layer override option [Y])
Outputs: a set of INFO-based image catalogs at $drs_loc/metadata/image_cats/info
Calls made: None

## 4.0 Primary DRS Update Operations

The Update DRS Application is triggered through a cron process (anticipated to be) on a nightly update schedule. At this point, the queue_update_job.pl program is executed which in turn calls the master driver for the application (drive_drs_update.pl). All programs are located at: $MOTHERSHIP/applications/update_drs.

Conceptually, the application operates in three stages for each DRS: 1) assemble desired conditions, 2) create and execute the workplan, and 3) update existing conditions. Every program module has its own potential points of failure, which are subject to internal tests within the code itself.

The heterogeneous nature of the programming environment required the development of an operating system level test for program failure, which is the creation of a system file (called "early_term") upon each program start up and its subsequent deletion when a program successfully exits. Some programs will not execute as long as the early_term file exists. This pre-test is strategically placed in the program execution sequence. The third logical section of the application (update existing conditions) represents a special case. It is essential for the DRS metadata to always reflect the actual conditions on the site. Therefore if the system fails part way through an update process, it is still essential to execute that section. The third section is intended to execute whenever the application proceeds into the second section.

By default, the Update DRS process operates selectively, for the most part only acting on the layers that appear in a final workplan. Exceptions are the processes which transfer the new layer def file, and AOI files to the target DRS, which are wholesale replacements.

Three programs have override switches which allow for a full site metafile rebuild: gen_layer_status.pl, update_drs_lib.aml, and update_image_cats.aml. In each case, a third optional argument of "Y" can be added. If this switch is not used, the processes will be guided by the desired_cond/action_layers file.

All temporary files are written to $prog_loc/desired_cond/$drsname/process. This prevents conflicts between concurrently running update processes.

The update_drs process as described here should never be applied to the main DRS (Mothership). Section 6.2 (below) provides the correct procedure for updating the main site.

## 5.0 DRS Update Reporting

The DRS update process spawns two types of reports: 1) general error report, and 2) site activity reports.

General error reports are funneled to an open file with a date stamp. All error activity reported during the course of a day is written to a single file. The application is very conservative, with most error conditions considered fatal, resulting in a update process abort with an error report section written to: $MOTHERSHIP/apps/update_drs/error_reports/[DRS site ID]_[data stamp]. Most error conditions are truly catastrophic, including: cannot connect to the DD, or a data_layer_def file is missing, or primary DRS files cannot be found. The exception is when a specific layer cannot be updated due to the absence of a DRS layer_status file which is noted in the error report while the process continues. In that situation, the layer in question should be unchanged on the DRS.

Activity reports are written to $MOTHERSHIP/apps/update_drs/desired_cond/ $drsname/act_reports, and are again stamped with a date. All activity for a given site during the course of a day is reported there. The file is initiated during the workplan triage process (triage_update_proc.pl) where statistics on update requests are generated. When "triage" reduces a workplan scope to conform to target site parameters, the purged data elements are listed here. The data copy process also contributes to activity_reports.

## 6.0 DRS Update Utilities

At the time of this writing the DRS site update process is supported by three other related applications that collectively constitute the full DRS site update environment. These include 1) DRS Site Installation, 2) Main DRS Metafile Rebuild, and 3) DRS Disrupted Operations Recovery. These are discussed in separate sections below.

### 6.1 DRS Site Installation

DRS site installation is performed by running *$MOTHERSHIP/apps/update_drs/install_drs.pl* with arguments of *$drsname* and *$drs_loc*. This simply establishes the minimum directory and file structures necessary for a DRS. After running this program, run *drive_drs_update.pl* (with the drsname argument) to actually populated the site with desired conditions.

### 6.2 Main DRS Metafile Rebuild

The main DRS (mothership) needs to have its metadata rebuilt regularly to reflect changes in data content. Metadata builds may be either "full" (meaning 'complete'), or "partial." This processes are similar to a regular site update, except that the data content of the site is not changed.

Full rebuilds are accomplished by running *drive_main_full.pl*. A program structure chart of this process is described in Figure 5. Full rebuilds usually take about 25 minutes to complete. They are the ultimate DRSMain recovery step and will disrupt DRSMain site usage within the last 12 minutes of the process. drive_main_full.pl takes the name "drsmain" as its sole argument.

Figure 5: DRSmain Full Build Process

```
drive_main_full.pl
      drive_desired_mother.pl
            make_desired_layer.pl*
            make_desired_aoi.pl*
      recov_from_update_abort.pl
            update_drs_def.pl*
            gen_layer_status.pl*
            update_drs_alyer_list.pl*
            update_drs_aoi.pl*
            Update_drs_lib.aml*
            Update_image_cat.aml*
```

Partial DRSMain rebuilds are more common and generally non-disruptive to users. They are accomplished by running *drive_main_partial.pl*, which has the following usage:

drive_main_partial.pl <DRSMain name> <list> <list of layers to update separated by spaces>

When called with only the first argument (currently, "drsmain"), the program will read the file: $MOTHERSHIP/apps/update_drs/desired_cond/<DRSMain name>/action_layers to obtain the list of layers to act on. When the keyword "list" is added, the application expects a space-separated list of layer names to follow. These arguments are used to create a new action_layers file, which is then used to fuel the process. A structure chart for this process is presented in Figure 6.

Figure 6: DRSMain Partial Build Process

```
drive_main_partial.pl
      drive_desired_mother.pl
            make_desired_layer.pl*
            make_desired_aoi.pl*
      existing_cond_driver.pl
            gen_layer_status.pl*
            update_drs_alyer_list.pl*
            update_drs_aoi.pl*
            Update_drs_lib.aml*
            Update_image_cat.aml*
```

## 6.3 DRS Disrupted Operations Recovery

It is sometimes necessary to manually disrupt a site update process[2]. The proper method for

[2]A "naturally" interrupted process, where the system fails gracefully from a recognized failure condition, will almost always continue and rebuild the target site, eliminating the need for taking the steps described here.

restoring a site to health depends on at which point in the process the interruption occurred. Termination at the desired conditions generation stage has no effect on the target site. However, once a data transfer process has begun, the content of the site has changed, and some decisions affecting remedial action must be made.

The user must be clear that there is no way to roll back the data changes that have occurred in an interrupted process. The only way to replace data is to verify the site Desired Conditions and restart the update process. The problem with simply restarting the process is that the new existing conditions have not been properly built, and if the site is not properly described internally, then the update process will not execute correctly.[3] Consider two scenarios: 1) The user interrupts a site update when they realize that some layers are being removed from the site accidentally, and 2) The user interrupts a site update when they realize that a copy process is going to take too long. In Scenario 1, if the process is simply restarted (after correcting the desired conditions), the deleted data will not be replaced, because that change is not reflected in the actual conditions. If the process is restarted in Scenario 2, all of the data that has been copied to date will be recopied again. Restoring the metafiles to reflect the actual conditions of a site is important.

Three categories of metafiles are affected here: 1) the layer registry (data_layer_def), 2) the area of interest files (aoi's), and 3) the layer_status files. The desired conditions and the existing conditions will have differences in these files.

The main question the Update DRS Application user must ask is whether the content of the site is best reflected in the desired conditions that were being executed, or the actual conditions that were present before site update began. To state it another way: do you want the site to go forward to reflect the partial update, or would you like the site layer description to be based on the previous set of conditions?

A partially executed workplan will have added or deleted data from the site. If a set of unwanted deletions has occurred, it is likely that the desired conditions were wrong and the user should seek recovery to the previous conditions. If the application was in the process of correctly copying new data to the site, and extensive deletion processes have not been conducted, then a site recovery based on the desired conditions would be most appropriate. Each option is described in subsections below.

### 6.3.1 Restoring DRS Metafiles Based on Previous Conditions

This is the best approach when data have primarily been subtracted from a site. To rebuild the DRS metafiles based on previous conditions, run the existing_cond_driver.pl program with the standard arguments.

### 6.3.2 Restoring DRS Metafiles Based on Desired Conditions

[3]Since the site update process is based on the comparison of desired versus actual conditions, an incorrect actual condition description will not provide expected results.

This is the best approach when there has been a net addition of data to the site. To rebuild DRS metafiles based on desired conditions, run the program recov_from_update_abort.pl with the standard arguments. This will read the data_layer_def file already on the site, and build the layer-specific metafiles *for the layers involved in the action*. Alternately, the administrator may wish to run recov_from_update_abort_full.pl to completely rebuild the site metafiles from scratch. This second option may take from 30-120 minutes to execute, and should not triggered during the daytime, as it will disrupt user access to the site. Both of these programs executes the sequence listed in Figure 7.

Figure 7: recov_from_update_abort program sequence

```
recov_from_update_abort.pl
           update_drs_def.pl*
           gen_layer_status.pl*
           update_drs_alyer_list.pl*
           update_drs_aoi.pl*
           Update_drs_lib.aml*
           Update_image_cat.aml*
```