

# Localization Overview

Briefly, the localization process is defined as automatically providing a minimally functional version of the site specific data required to support operations at a local office based on a single national configuration data set. The localization process provides the means to override that default functionality locally.

Most everything that has to do with localization lives in the directory `${FXA_HOME}/data` or its subdirectories. `FXA_HOME` is an environment variable that currently points to `/awips/fxa` by default. Assume that `LLL` is an arbitrary localization ID (examples of this would be `EAX` and `BOX`, which refer to the WFOs at Pleasant Hill, MO and Taunton MA, respectively). Here are the directories that are important to localization.

<code>\${FXA_HOME}/data/</code>	Contains a few of the national configuration data set files.
<code>\${FXA_HOME}/data/localization/nationalData/</code>	Contains the rest of the national data set files. This directory is cross mounted to the data device.
<code>\${FXA_HOME}/data/localization/LLL/</code>	Contains the files specific to the <code>LLL</code> site that are delivered with each software release.
<code>\${FXA_CUSTOM_FILES}/</code>	Contains locally installed customizations that are not delivered with each software release. This currently defaults to <code>/data/fxa/customFiles</code> .
<code>\${FXA_HOME}/data/localizationDataSets/LLL/</code>	When running a localization for site <code>LLL</code> , files get written to this directory. Files in this directory are read at runtime and make the behavior of the <code>LLL</code> localization unique.
<code>\${FXA_HOME}/data/localization/scripts</code>	This directory contains all of the localization scripts.
<code>\${FXA_HOME}/data/localization/documentation</code>	This directory contains online documentation about localization.

Please note that the contents of the directories `${FXA_CUSTOM_FILES}/` and `${FXA_HOME}/data/localization/LLL/` cannot affect the way the workstation or ingest behave, they can only change the way a localization runs. The directories that can affect the way AWIPS process behave are `${FXA_HOME}/data/`, `${FXA_HOME}/data/localization/nationalData/` and `${FXA_HOME}/data/localizationDataSets/LLL/`.

One should also note that any change made manually in the directory `${FXA_HOME}/data/localizationDataSets/LLL/` can potentially be overwritten the next time a localization is run; thus this is the wrong place to make changes to the default functionality. This not only refers to modifying existing files, but changes implemented by adding or removing files. It is reasonable to do testing by making manual modifications in

`${FXA_HOME}/data/localizationDataSets/LLL/`, but one should never attempt to permanently implement a change this way.

An even more important corollary to this is *never make a file in the localization data set unwritable in an effort to preserve it*. This can cause the localization to break. There have been instances where an attempt to distribute new national data sets has been disrupted by widespread presence of unwritable files. Because of this, localization starts by making all of the files in the localization data set writable. If one really needs to modify a file that localization has produced, it is better to create a script patch file with sed, awk, or perl commands to modify the file. See section 2 of [fileChanges](#) or [scriptOverride](#) for more information about script patch files.

Changes in `${FXA_HOME}/data/` or `${FXA_HOME}/data/localization/nationalData/` will not be overwritten by running a localization, but they will be overwritten when a new build is released. Furthermore, they can inadvertently affect other localizations that an office might need to support backup responsibilities. Thus, implementing changes here is also not recommended.

Changes in `${FXA_HOME}/data/localization/LLL/` are not guaranteed to survive an upgrade, but they will not be overwritten by running localizations and will not affect other localizations. This is often the best place to implement a change, but one needs to save any files placed here in case they need to be restored after a new build is delivered.

The best place to implement a change is in `${FXA_CUSTOM_FILES}/`. A file here that begins with ``LLL-'` will be specific to a particular localization; without that prefix it can potentially affect all localizations. Files here will survive both running localizations and the installation of a new build. However, not every override file can be implemented in this directory. One should see the on-line localization document [fileChanges](#) for more information about which override files can be implemented where.

There are some very important things to keep in mind when making local modifications to AWIPS. First, don't change things on the data servers unless you are instructed to. A corollary to this is to make sure you try changes on one workstation before you propagate them to the others. Second, if applicable, always make a copy of the file you are modifying in case you need to back out of the change. Third, always keep a copy of your new file in case you either have to restore it after an upgrade or just to back it up in case you have a disk failure or something.

The script `${FXA_HOME}/data/localization/scripts/mainScript.csh` is what is actually executed to run a localization. Running the script with a single argument ``h'` will print a usage message.

---

# Documentation

There is a known limitation concerning this documentation: a good deal of it was written in order to support other developers that might have to maintain localization code and thus it is reference rather than tutorial in nature. That being said, there is still a lot of information that one can glean from these documents. Furthermore, using the online documents delivered with the load reduces the chance that a person would be working with information not applicable to the current software build.

Here is a catalog of the localization documentation. Where documentation files refer to utility programs, the name of the program is the same as the name of the documentation file without the extension, and the programs can be found in `${FXA_HOME}/bin`.

## Descriptive Documents

### [localization](#)

This is the most detailed and complete document concerning localization. Definitely not light reading.

### [adaptivePlanViewPlotting](#)

Information about how to change the look and feel of existing plan view plot displays and how to add new ones.

### [characterSets](#)

This contains a high level treatment of AWIPS character sets, including illustrations.

### [developers](#)

Originally written to help developers make the transition from pre-localization environment to the post-localization environment, this is a useful high level treatment of how to run localizations.

### [directives](#)

Information about how to manipulate software switches that appear in the `mainConfig.txt` and `wwaConfig.txt` files. Includes a useful high-level treatment of file override.

### [families](#)

This document discusses the mechanism by which families are auto-generated in the localization.

## [fileChanges](#)

Contains a fairly detailed treatment of how file override works. It also contains tables that are a comprehensive list of all of the files in localization that can have their functionality overridden, in which directories that override can be implemented, and which localization tasks need to be run to complete the implementation.

## [gridTables](#)

Information about how to manage the tables that control what products are available through the Volume Browser.

## [ldadContouring](#)

Information about how to configure contour depictions of arbitrary LDAD mesonet variables.

## [mainScript](#)

The usage message printed by mainScript.csh.

## [purgeTables](#)

This document discusses how the purger configuration tables work.

## [radarLocalization](#)

General information on radar localization, with some emphasis on new OB3 features.

## [radarMosaics](#)

Describes how to manage the radar mosaics that are generated on the fly.

## [README.GRIDS](#)

Contains a high-level treatment of managing the metadata associated with the Volume Browser. Very developer oriented, it should be used with care.

## [satDirs](#)

This is an overview of the default satellite data directory structure on AWIPS.

## [scriptOverride](#)

Explains how overriding the functionality of localization scripts works. This document was written because the way script override works changed significantly in OB5.

## [shapeFileDisplay](#)

This describes how one can display shape files directly as map backgrounds.

## [staticProgDisc](#)

Information about how the visibility of stations in plan view plots is controlled as one zooms in.

## [styleRules](#)

Information about how to change the look and feel (for example, contour intervals) of Volume Browser products.

## [TextTemplate](#)

Information about how to change the functionality of warnGen templates.

## [warnGenBackup](#)

A very high level description of how to set up service backup for warnGen.

## **Utility Programs**

### [bcdProc](#)

Usage documentation for the utility program that manipulates binary cartographic data files (.bcd and .bcx files), which are the primary real time source for vector map background data.

### [fileMover](#)

Usage documentation for the utility program fileMover. This utility program is used nearly every time a file is moved into the localization data set from some other directory, instead of a mv or cp command. The way the utility program fileMover works is central to making file override work correctly.

### [GELTtest](#)

Usage documentation for the utility program that is a unit test for the software that reads geographic entity lookup tables. It is also used in localization scripts to extract information from GELTs.

### [image\\_mask](#)

Usage documentation for the utility program that is used to manipulate the contents of one GELT based on another.

### **[initCdlTemplate](#)**

Usage documentation for the utility program that writes geographically dependent and topographic information into a template netCDF file.

### **[keyMunge](#)**

Usage documentation for the utility program that is used to automatically change key values in any AWIPS key indexed file.

### **[makeGridKeyTables](#)**

Usage documentation for a program that builds all of the keys for the Volume Browser based on the source, field, and level tables.

### **[maksubgrid](#)**

Usage documentation for the utility program that makes geographic depictor files that represent a clipped portion of a grid.

### **[maksuparg](#)**

Usage documentation for the utility program that creates geographic depictor files (.sup files).

### **[makthermo](#)**

Usage documentation for the utility program that makes thermodynamic depictor files.

### **[makxsect](#)**

Usage documentation for the utility program that makes cross section depictor files.

### **[newGELTmaker](#)**

Usage documentation for a utility program that creates geographic entity lookup tables (GELTs). This program is a more flexible and user friendly version of the older program makeGeoTables (this program is obsolete and has been removed from AWIPS). It also has some useful overview information about GELTs.

### **[pasteUtil](#)**

Usage documentation for the utility program that merges several files together, interleaving the data as columns.

### [processStyleInfo](#)

Usage documentation for a program that creates all of the style information for Volume Browser products based on .rules files.

### [rangeAzimuth](#)

Usage documentation for the utility program that is used to do range and azimuth calculations in scripts.

### [reformatTest](#)

Usage documentation for a utility program that was written primarily to convert older netCDF point data files and convert them to the new format required to support adaptive plan view plotting. This program can also be used to create and modify such data sets for testing.

### [shp2bcd](#)

Usage documentation for the utility program that reads data from shape files and writes out binary cartographic data files.

### [testDepictorTable](#)

Usage documentation for the utility program that makes depictor tables, which are used to optimize the use of two geographic depictors in remapping operations.

### [testFileNotify](#)

Usage documentation for a utility program that allows one to generate notifications for data sets not brought in through the normal AWIPS ingest.

### [testGridKeyServer](#)

Usage documentation for a program that is a unit test for the software that reads in source, level, and field tables for the Volume Browser. It is also used in localization scripts to extract information from these tables.

### [testGridSliceWrapper](#)

Usage documentation for a program allows access to AWIPS gridded data sets. Returns data as plain ASCII.

### [test\\_grhi\\_remap](#)

Usage documentation for the utility program that is a standalone gridded data to image data remapper.

### **testPlotDesign**

Usage documentation for a utility program that can be used to test and verify the parsing of design files. Can also test the inventory and data access functions.

### **textBufferTest**

Usage documentation for the utility program that can read a text file, interpreting all of the #include commands and removing comments and blank lines.

### **va driver**

Primarily usage documentation about the utility program which is used to create progressive disclosure information for plan view plots. Includes a useful treatment of the files involved in this process.

---

# The WFO-Advanced Localization Process

---

## Table of Contents

- [1. What is localization?](#)
- [2. Some background](#)
- [3. Overview](#)
  - [3.1\) The Localization Software Environment](#)
  - [3.2\) Defining Localizations](#)
  - [3.3\) Creating and Using Localizations](#)
- [4. Functional breakdown - Localization Scripts](#)
  - [4.1\) mainScript.csh](#)
  - [4.2\) Some generic utility scripts](#)
  - [4.3\) makeDataSups.csh](#)
  - [4.4\) makeScales.csh](#)
  - [4.5\) makeClipSups.csh](#)
  - [4.6\) assembleTables.csh](#)
  - [4.7\) makeTextKeys.csh](#)
  - [4.8\) makeTopoFiles.csh](#)
  - [4.9\) updateGridFiles.csh](#)
  - [4.10\) updateRadarFiles.ksh](#)
  - [4.11\) makeMapFiles.csh](#)
  - [4.12\) makeWWAtables.csh](#)
  - [4.13\) makeStationFiles.csh](#)
  - [4.14\) makeDirectories.csh](#)
  - [4.15\) createAuxFiles.csh](#)
  - [4.16\) makePurgeTables.csh](#)
  - [4.17\) fxatextTriggerConfig.sh](#)
  - [4.18\) LAPS localization](#)
  - [4.19\) MSAS localization](#)
- [5. Files in the national data set](#)
  - [5.1\) Scale table](#)
  - [5.2\) Process config files](#)
  - [5.3\) Main and manual data and depict key files](#)
  - [5.4\) Text database and depictable localization](#)
  - [5.5\) Satellite data and depictable keys](#)
  - [5.6\) Product button file](#)
  - [5.7\) Menu files](#)
  - [5.8\) Shape files](#)
  - [5.9\) Vector map background files](#)
  - [5.10\) Topography files](#)

- [5.11\) Gridded data and Volume Browser tables](#)
- [5.12\) Display style files](#)
- [5.13\) Radar-related files](#)
- [5.14\) Files related to WarnGen](#)
- [5.15\) Files that control stochastic progressive disclosure](#)
- [5.16\) Color table files](#)
- [5.17\) Run time config files](#)
- [5.18\) Files which control plan view plotting](#)
- [5.19\) Files that specify the MSAS domain, background, and user-definable variables](#)
- [6. Other site-specific control files](#)
  - [6.1\) Localization config files](#)
    - [6.1.1\) LLL-mainConfig.txt](#)
    - [6.1.2\) LLL-cwa.asc](#)
    - [6.1.3\) LLL-wwaConfig.txt](#)
  - [6.2\) Files referred to through #include](#)
    - [6.2.1\) Depict keys, data keys, and product buttons](#)
    - [6.2.2\) Menu files](#)
  - [6.3\) File override expansion](#)
    - [6.3.1\) Menu files](#)
    - [6.3.2\) Map background files](#)
    - [6.3.3\) Topography and cdl files for gridded data sources](#)
    - [6.3.4\) Product templates for WarnGen](#)
    - [6.3.5\) GELT script files](#)
    - [6.3.6\) Station lists](#)
  - [6.4\) Files used to activate and deactivate data sets](#)
- [7. Localization programs](#)
  - [7.1\) fileMover](#)
  - [7.2\) Programs that create depictor files](#)
  - [7.3\) Programs that manipulate key files and menu files](#)
  - [7.4\) Programs that manipulate cartographic data sets](#)
  - [7.5\) Programs that manipulate tables for gridded data and the Volume Browser](#)
  - [7.6\) Program that creates geographic entity lookup tables](#)
  - [7.7\) Programs that create station and localization lists](#)
- [8. Realization definition files](#)
- [9. Customization](#)

## 1. What is localization?

The WFO-Advanced workstation and data ingest software have been installed in every National Weather Service (NWS) Weather Forecast Office (WFO), River Forecast Center (RFC), and Regional Headquarters in the nation, as well as at some national centers. While a great deal of the system functions and data sets are common to nearly all of these offices, much will be unique to each site. The localization process is defined as automatically providing a satisfactory version of the site specific data required to support operations at a local office based on a single national configuration data set.

The form of the national configuration data set has been designed in a joint effort between the NWS and FSL, and is still under development. The national configuration data set currently contains information about counties, cities, zones, topography, rivers, etc., and is maintained at NWS headquarters in cooperation with the local offices. Changes to these basic data sets will be distributed to each office. Each office is able to incorporate these changes into all of their local configuration data sets by running a single script.

This document was originally written with other developers as a target audience. If one is new to the world of localization, it's best to first read the [Localization Overview](#).

## 2. Some background

When the WFO Advanced system was first developed, all of the configuration data sets were treated like source code; that is, they were manually maintained and kept in a source code control system. As the complexity of the configuration data sets increased, it became impractical to maintain each individual configuration data item manually. This meant that for some configuration data sets, we began to maintain tables that described how to construct these data sets, and created these configuration data sets by running programs. The gridded data sets accessible through the Volume Browser were the first to use this paradigm. These dependent configuration data sets were initially still maintained in source control.

The real beginnings of the localization process (although at the time it was not called that) were when we removed these dependent configuration data sets from source control and began generating them on a regular basis in our software builds. What this did, in effect, was to generate a default Denver localization each time we did a software build. This created a problem in that our software directory structure was designed to handle the interrelationships in our source code well, but it did not well reflect the interrelationships between our dependent data sets and the programs that created them. As such the data targets in our software builds got to be very messy, so some sort of separate process for generating the dependent data sets would have eventually been developed even if national deployment was not an issue.

When it became known that the WFO-Advanced system was going to be deployed nationwide, it was immediately apparent that developing an automated localization process independent of

our software build process was absolutely essential. Design for this started in October 1996 and development began in December that year. By mid January 1997, enough of the localization process was working that we were able to remove the default Denver localization from the software build process. In late January the first of the data sets provided by NWS headquarters were incorporated into the localization process. The WFO-Advanced system that went to the AMS conference in February 1997 was the first deployment of a WFO-Advanced system that relied on the localization process for its configuration data. The WFO-Advanced upgrade installed in the Denver WFO at the end of February 1997 was based on the localization process, as was the first major software hand-off the NWS in March 1997. In October 1997, a formal methodology for including local customization was developed. As of this writing, the basic structure of the localization scripts is complete, and all of the major data sets that make up the national configuration data set are implemented.

### **3. Overview**

#### **3.1) The Localization Software Environment**

The localization process in WFO Advanced relies on a national configuration data set, a set of scripts and programs for creating the local configuration data sets from the national data, and a set of files that allow one to tailor a specific localization beyond what would be available from the national configuration data set and the default behavior of the localization scripts.

In our source code control system in `$FXA_HOME/src`, there is a subdirectory called 'localization' where all of the scripts, national data, and local data used by the localization process reside. Source code for programs used by the localization scripts is distributed amongst the other source directories; the localization directory contains only scripts, configuration data, and documentation.

There are five subdirectories in the localization source directory: 'scripts,' 'nationalData,' 'documentation,' 'localData,' and 'realizations.' As the names would suggest, the scripts directory is where the localization scripts reside, the nationalData directory is where the national configuration data set resides, and the documentation directory is where user and developer documentation is kept. The localData directory is where the site-specific control files are kept, and the realizations directory is the home of special site-specific control files that change the behavior of groups of localizations.

Running the 'data' target in the localization source directory causes all of the files needed to create localizations to be moved to the directory `$FXA_HOME/data/localization`. When localizations are created, they use the data and scripts from `$FXA_HOME/data/localization`, not from the source tree. This is because it is necessary to create localizations in the field, and field sites will not have a source tree, only `$FXA_HOME/bin` and `$FXA_HOME/data`. It is important to remember that the act of running the data target in the localization source directory only moves files over to the directory `$FXA_HOME/data/localization`; it does not create a usable localization.

Upon running the data target in the localization source directory, the files in the nationalData, scripts, and documentation subdirectories get moved directly to subdirectories of the same name in the directory \$FXA\_HOME/data/localization.

The files in src/localization/localData get exploded into a directory structure. The name of every site-specific control file in localData looks like LLL-blahblah.blah, where LLL is the identifier of a localization (not necessarily 3 characters long). A file in src/localization/localData called LLL-blahblah.blah ends up being moved to the directory named data/localization/LLL by the data target in the localization source directory. The reason for this is that while it is painful to add directories in our source code control system, we want what is installed in the field to have a separate site control directory for each localization. Because a '-' is used as the delimiter that separates the localization identifier from the rest of the file name for site specific control files, one should never try to use a '-' in a localization identifier.

Files in src/localization/realizations behave similarly. The name of every realization file in realizations looks like RRR--blahblah.blah, where RRR is now the realization identifier. Such a file ends up in data/localization/realizations/RRR. Two '-' characters are used as the delimiter so it is harder to mistake a realization file from a local site specific file. Realizations are mentioned here so that the reader will understand their general function. Not much else will be said about them until much later because most localizations are not associated with a realization.

In order to make use of the results of the localization process, all of the WFO Advanced workstation and data ingest processes use a single software module, called InfoFileServer, to locate static metadata files. This module always checks a series of predefined directories, some specific to a given localization and some generic, in a well-defined order when trying to locate a file. Incorporated into most of the software that actually reads static metadata files is the ability to interpret C/C++ style #include statements, along with automatic environment variable translation in these statements. This means that any new software written that reads static metadata files should always use the InfoFileServer class to locate these files.

All of the discussion in the next two sections is predicated on the user having access to a full build of the WFO Advanced software tree.

### **3.2) Defining Localizations**

So far, localization identifiers have been discussed in only generic terms. It is not possible to just pick a character string at random and use it as a localization identifier -- a localization identifier must first be defined.

The NWS has provided us with a file that contains information about county warning areas. The identifiers of all county warning areas are automatically defined to be valid localization identifiers. One can produce a list of these identifiers by running the command

```
$FXA_HOME/data/localization/scripts/cwalds.csh
```

Running this same command with the -a option will provide a list of all currently defined localizations.

It is also possible to define additional localizations by creating certain types of site-specific control files. As the reader will recall from the previous section, site-specific control files exist in the source tree in the directory `src/localization/localData`, have file names that look like `LLL-blahblah.blah`, and get moved to `data/localization/LLL` when the software tree is built. A localization can be defined with a site-specific control file that is named either `LLL-mainConfig.txt` or `LLL-cwa.asc`. The `cwa.asc` file is just a one line file with a latitude and longitude around which to center the WFO scale. Files with names that look like `LLL-blahblahConfig.txt` contain what are referred to as *directives* (see [directives.html](#)), which are lines in a file that begin with `@@@`. In order for a `mainConfig.txt` file to properly define a localization, it must contain a `@@@WFO` directive, a `@@@CLONE` directive, or a `@@@REALIZATION` directive. The argument of a `@@@WFO` directive must be a county warning area identifier, and the argument of a `@@@CLONE` directive must be the identifier of some other defined localization which is not itself a clone (see also [Section 6.1](#)). The argument of a `@@@REALIZATION` directive must be the identifier of a correctly defined realization (also see [Section 8](#)).

Here we have mentioned only two possible types of site-specific control files and two possible directives. There will be much more on this later.

### 3.3) Creating and Using Localizations

To actually create a localization, one also needs to define some environment variables. Here is a list of the environment variables that are currently applicable to the localization process, along with their values:

```
FXA_NATL_CONFIG_DATA = ${FXA_HOME}/data/localization
FXA_LOCALIZATION_SCRIPTS = ${FXA_HOME}/data/localization/scripts
FXA_LOCALIZATION_ROOT = ${FXA_HOME}/data/localizationDataSets
FXA_LOCAL_SITE = LLL
FXA_INGEST_SITE = III
```

`FXA_NATL_CONFIG_DATA` points to where the source data for localizations reside, and

`FXA_LOCALIZATION_ROOT` points to where the finished localization subdirectories go.

`FXA_LOCALIZATION_SCRIPTS` is where all of the localization scripts are run from.

`FXA_LOCAL_SITE` is meant to point to the localization currently being used, and

`FXA_INGEST_SITE` is meant to point to the localization currently being run on the data server.

There is also a bare minimum set of programs that must be built in order for one to run all tasks of a localization. Programs not needed to run the default list of tasks are noted with an asterisk. The function of these programs will be described in detail later. For now, the list of programs is:

```
${FXA_HOME}/src/dm/shapefile/shp2bcd
${FXA_HOME}/src/dm/grid/initCdlTemplate
${FXA_HOME}/src/dm/grid/makeGridKeyTables
${FXA_HOME}/src/dm/grid/processStyleInfo
```

```
$FXA_HOME/src/dm/grid/testGridKeyServer
$FXA_HOME/src/dataMgmt/fileMover
$FXA_HOME/src/dataMgmt/keyMunge
$FXA_HOME/src/dataMgmt/pasteUtil
$FXA_HOME/src/util/textBufferTest
$FXA_HOME/src/geoLib/bcdProc
$FXA_HOME/src/geoLib/maksuparg
$FXA_HOME/src/geoLib/test_grhi_remap
$FXA_HOME/src/geoLib/maksubgrid
$FXA_HOME/src/geoLib/rangeAzimuth
$FXA_HOME/src/mapping/testDepictorTable
$FXA_HOME/src/mapping/makthermo
$FXA_HOME/src/mapping/makxsect
$FXA_HOME/src/mapping/testDepictorTable
$FXA_HOME/src/staticPlotData/va_driver
$FXA_HOME/src/staticPlotData/masterToGoodness
$FXA_HOME/src/geoTables/newGELTmaker
$FXA_HOME/src/geoTables/GELTtest
$FXA_HOME/src/geoTables/image_mask
$FXA_HOME/src/dm/point/reformatTest
$FXA_HOME/src/dm/point/testPlotDesign
$FXA_HOME/src/tstorm/localize/create_radarLoc *
$FXA_HOME/src/tstorm/localize/sitefinder *
$FXA_HOME/src/ffmp/localize/localizeForFFMP *
```

In an environment with no source tree, all of these programs will, of course, be in \$FXA\_HOME/bin.

In order to actually create a localization, go to the directory \$FXA\_HOME/data/localization/scripts and issue the command 'mainScript.csh LLL LLL'. This will build the localization with the identifier LLL. mainScript.csh takes 5-20 minutes to run, depending on the type of machine one is running on and the complexity of the localization. It will produce quite a few diagnostics, which will be discussed later. The files for this localization are put in a directory named \$FXA\_HOME/data/localizationDataSets/LLL.

To run a workstation or an ingest process, the environment variable FXA\_LOCAL\_SITE must be set to the identifier of a successfully created localization. This is because there is a single software module used throughout the WFO Advanced software to find static metadata files. When asked to find a file, it first looks in \$FXA\_LOCALIZATION\_ROOT/\$FXA\_LOCAL\_SITE, then in \$FXA\_NATL\_CONFIG\_DATA/nationalData, then in \$FXA\_HOME/data, and if all these fail it will try the current working directory.

The environment variable FXA\_INGEST\_SITE is important because it is possible for a data server to be running one localization when a workstation is running a different one. In general, the data server will run the same localization over time, where workstations might run different localizations. If, when building a localization, one issued the command 'mainScript.csh LLL III', this will build a localization named LLL on the assumption that its data server is using the localization named III. Issuing the command 'mainScript.csh' is the same as issuing the command 'mainScript.csh \$FXA\_LOCAL\_SITE \$FXA\_INGEST\_SITE', and the command

'mainScript.csh LLL' will produce the same result as running the command 'mainScript.csh LLL \$FXA\_INGEST\_SITE'. What this means is that FXA\_INGEST\_SITE is set to be whatever localization the data server is running, so that when localizations are built using a single localization identifier, they will be able to properly interpret data from the data server.

## 4. Functional breakdown - Localization scripts

As previously described, all of the localization scripts are placed in the software tree in the directory \$FXA\_HOME/src/localization/scripts, and run out of the directory pointed to by the environment variable FXA\_LOCALIZATION\_SCRIPTS. FXA\_LOCALIZATION\_SCRIPTS is currently defined to be \$FXA\_HOME/data/localization/scripts.

### 4.1) mainScript.csh

The script mainScript.csh is mostly an executive; it delegates the creation of localization data sets to subordinate scripts. mainScript.csh verifies that the correct localization environment exists and that the localization that the user is attempting to build is viable. mainScript.csh also performs a couple of functions that are hard to isolate in a single subordinate scripts; assuring that any out-of-date mapping tables are disposed of and removing any files created in the localization data set that are identical to files in \$FXA\_HOME/data or nationalData/. It also allows the user to select which portions of the entire localization function are to be performed.

The usage of mainScript.csh is as follows:

```
mainScript.csh {h} {n} {f} {t} {v} {+task} {-task} {-task} {loc_id} {ingest_id}
```

Note that all of the arguments are optional. However, if run with no arguments at all in an environment without a source tree, it will print out a usage message. The usage message can also be printed out by using the 'h' argument. If run with no arguments with a source tree present, mainScript.csh will perform all default localization tasks with the localization identifier of \$FXA\_LOCAL\_SITE and an ingest localization identifier of \$FXA\_INGEST\_SITE. The 'loc\_id' argument allows one to specify the localization identifier on the command line, and the 'ingest\_id' argument allows one to specify the ingest localization identifier on the command line.

The 'n' option must be used if one is changing the customization environment for an existing localization. This means when the value of the environment variables FXA\_CUSTOM\_FILES or FXA\_CUSTOM\_VERSION changes. See section 9.0 on [customization](#) for more information about this. The 'n' flag must also be used if one wants to rerun an existing localization using a different ingest site.

There is now some logic in localization that allows it to detect when certain files are up to date and thus avoid recreating those files. If the 'f' option is present, this logic is disabled.

When mainScript.csh runs, it will echo to the user the list of localization tasks it is performing. Also, it will tell the user which subordinate script it is currently running, and some of the subordinate scripts produce some diagnostics of their own. The 'v' option will cause it to echo individual commands executed in the subordinate scripts. Currently, the complete list of default task options is as follows:

dataSups scales clipSups tables text topo grids radar maps wwa station

The '+task' option means perform that task and any tasks that follow. The '-task' option means just perform that task. One should use the +task option only once and it should be the first task option. One can use as many -task options as needed. Using the option '+dataSups' would be the same as the default behavior. To just verify whether a localization is viable, one can use a -task option for a non-existent task, such as '-x'. There are also seven non-default tasks, all of which but the first two run after all the default ones. They are invoked with one of the following list of task IDs:

laps msas dirs auxFiles scan purge trigger

The 'laps' task is used to create metadata specifically for running the Local Analysis and Prediction System. Likewise, the 'msas' task creates metadata for running the MAPS Surface Assimilation System (MSAS). The 'dirs' task will assure the creation of all data directories on \$FXA\_DATA, as determined by the current state of the dataInfo.txt file, with all of its include files. 'auxFiles' will create any other miscellaneous files that need to be moved to the data device. The 'scan' task creates metadata for running SCAN/FFMP. The 'purge' task will build the purge tables for the new purger. 'trigger' creates text product triggers.

A task option of '-all' will cause all default and non-default tasks to run. Additionally, the arguments '-WS', '-DS', and '-PX' will result in running only those tasks absolutely necessary for localizations that reside on the workstation, data server, and application server, respectively. The task -WWA (upper case as opposed to lower case) will run just enough localization tasks to support running WarnGen for a localization ID that has not yet been run. A leading 't' option will cause it to only verify that the localization ID selected is valid, list the tasks that would be run, and verify the path to each of the subordinate scripts being used.

The task selection functionality of mainScript.csh would be subverted should the user choose to define localization identifiers that begin with a '-' or '+', so this should never be done.

The user should note that the task options cannot change the order in which the subordinate scripts are run; it can only specify which subordinate scripts are run. This is because some tasks are dependent on previous tasks to function correctly. In general, one can expect to get meaningful results from running a given task only when the all of the preceding tasks have been successfully completed.

Within mainScript.csh it will also try to run subordinate scripts called test1.csh, test2.csh, or test3.csh if the task options '-test1', '-test2', or '-test3' are supplied. The user can place a test script by any of these names into \$FXA\_LOCALIZATION\_SCRIPTS and have them run within the localization environment. These tasks always run after the regular tasks have completed, and

will not be activated by the '-all' task option. Another task that behaves just like the test tasks is (run last, not with -all) is the 'fixGeo' task. Running this task on the data server will remedy the situation where the template files in the gridded data directories do not contain the correct geographic information.

The environment variable `FXA_LOCALIZATION_LOG`, if set, will cause the diagnostic output from `mainScript.csh` to be written to that file. Also, this will cause additional diagnostic output to be generated, including lists of files changed since the last time a localization was run. Using the 't' or 'v' flags will cause logging to be turned off.

Sections 4.3 through 4.14 will talk about all of the subordinate scripts in turn.

## **4.2) Some generic utility scripts**

There are three generic utility scripts that are used by virtually every subordinate script to help do their work. `getPath.csh` is a generic file finding script that is very analogous to the module `InfoFileServer` found in `$FXA_HOME/src/foundation`. `fileGrab.csh` is used to move groups of site specific control files with similar names from `$FXA_HOME/data/localization` to their proper place in `$FXA_HOME/data/localizationDataSets`. `doPatches.csh` is actually meant to be sourced rather than run as an independent script. In a subordinate script named `blahblahblah.csh`, sourcing `doPatches.csh` will cause the site specific control file `LLL-blahblahblah.patch` to be sourced if it exists. This allows an easy way to add functionality to subordinate scripts that cannot be done with a file replacement operation. There is also an analog to `doPatches.csh` called `doPatchesI.csh` that can be sourced to do the same thing based on the ingest localization instead of the display localization. See [scriptOverride.html](#) for more information on overriding script behavior.

There are four other generic utility scripts that are not as widely used, but nonetheless are very important: `configValue.csh` is used to obtain the value of directives; `newerUtil.csh` provides a means to determine if a dependent file is older than the file from which it is created, and thus must be recreated; `scaleSup.csh` is used to get the geographic information file (depictor file) for a given scale index; and `stdErr.ksh` gives a unified way of sending text to standard error -- this is the only way to produce diagnostics in scripts that output data to standard output as their primary function.

## **4.3) makeDataSups.csh**

The script `makeDataSups.csh` is controlled by the 'dataSups' task option. The purpose of this script is to create all of the geographic information files (depictor files) that describe the geographic characteristics of various data sets. In order to do its job, this script makes use of the programs `maksuparg` and `makthermo` and the script `raobUtil.csh`.

## **4.4) makeScales.csh**

The script `makeScales.csh` is controlled by the 'scales' task option. The purpose of this script is to create all of the geographic information files (depictor files) that describe the geographic characteristics of displays. This includes display scales, of course, but also includes the default state of the movable cross section, time section, and sounding depictors. It is also where the `scaleInfo.txt` file is handled. In order to do its job, this script makes use of the programs `maksuparg`, `makxsect`, `makthermo`, and `rangeAzimuth`. This is the only subordinate script where the generic `.patch` file replaces functionality instead of adding functionality when available and sourced.

Unlike the other subordinate scripts in the localization process, this one has an optional command argument. The optional argument is meant to be a localization identifier. When the optional argument is supplied, the depictor files created for each scale will be tagged with that localization identifier instead of named generically.

#### **4.5) `makeClipSups.csh`**

The script `makeClipSups.csh` is controlled by the 'clipSups' task option. When the 'clipSups' task is invoked, `makeScales.csh` is first called with the ingest localization identifier (`$FXA_INGEST_SITE` by default) as an argument. This creates a set of scale depictors that are specifically identified as being associated with the localization that is assumed to be running on the data server. Then, `makeClipSups.csh` is called, which actually generates those scale dependent data set depictor files that are other than just tagged scale depictors, such as the one for the clipped mesoEta grids. This is necessary because some data sets are created in a manner that is dependent on a display scale, such as the regional satellite sector, the mesoEta grids, or the LAPS grids. If a workstation were running a different localization than the data server and only had access to its own generically named depictor files, it would attempt to map these clipped data sets as if they were clipped according to its own localization, not the data server's. The user should note that it is within this second invocation of `makeScales.csh` that the decision is made as to whether to use east or west satellite.

#### **4.6) `assembleTables.csh`**

The script `assembleTables.csh` is controlled by the 'tables' task option. This script performs many functions. It sees to it that files containing the manually defined data and depictable keys are in place, and creates keys for satellite products. Satellite keys are adjusted for east vs west CONUS sectors and to use the regional clip area of the data server through the inclusion of a depictor file specifically tagged with the ingest site localization ID. `assembleTables.csh` grabs files containing locally defined data and depictable keys as well. For product buttons, it also verifies that the default manually defined version of that table is in place and grabs the file with locally defined buttons, if needed. For menus, it makes use of the `raobUtil.csh` script to construct a local RAOB menu with about 5 to 10 sites close to the center of the localization, and to place the 2 to 4 closest RAOBs directly on the Upper Air menu. It sees to it that the default menus are in place and will move any locally defined menus into the localization data set. `assembleTables.csh` is where file override is implemented for design files and lookup table files,

which control plan view plotting (see [adaptivePlanViewPlotting.html](http://adaptivePlanViewPlotting.html) for more info). This script is also where color tables are handled.

#### **4.7) makeTextKeys.csh**

The script makeTextKeys.csh is controlled by the 'text' task option. Using the textUtil.csh script, makeTextKeys.csh determines the proper AFOS CCC to use and places it in the fxa.config file. It also does distance based assignment of the number of versions of certain text products to keep, and automatically generates data and depictable keys for text depictables that only use some regional or local text products.

#### **4.8) makeTopoFiles.csh**

The script makeTopoFiles.csh is controlled by the 'topo' task option. This script makes the topography data for high resolution topographic images and for those gridded data sources whose geography is dependent on the location of the display scales. This script has 2.5 minute world topography and 1km US topography at its primary data sets to derive other topography data from. It also has 1km Alaska, 30 second Pacific and 30 second Caribbean data sets available. For creating certain data sets, the maksuparg program is used to determine whether the area in question overlaps the US, Pacific, Alaska, or Caribbean topography, otherwise it will use the world topography. The program test\_grhi\_remap is what is used to make the derived data sets. Topography data for gridded data sources can also be made available by moving \*.topo files from site specific control files into the localization data set.

#### **4.9) makeGridSourceTable.csh and updateGridFiles.csh**

The scripts makeGridSourceTable.csh and updateGridFiles.csh are controlled by the 'grids' task option. These two scripts collectively perform all the tasks necessary to make gridded data available through the Volume Browser. The script makeGridSourceTable.csh, whose job it is to create the final version of gridSourceTable.txt, generally runs before any other task because other tasks need that file. (The one exception to this rule is that the 'msas' task, if selected, will run before the 'grids' task to create the files that define the MSAS domain.) The file gridSourceTable.txt defines some characteristics for gridded data sources and specifies which ones are currently active.

The script updateGridFiles.csh is used to create template files from the .cdl files for active gridded data sources, and build the depictable keys and data keys associated with gridded data. updateGridFiles.csh also creates style information (for example, contour intervals) from style rules files, generates Volume Browser selection menus for sources and cross sections, and generates the depictor files that represent the predefined lat/lon cross section baselines. updateGridFiles.csh uses the program testGridKeyServer to generate lat/lon baselines and Volume Browser menus, the program ncgen to make .cdlTemplate files, the program initCdlTemplate to place geographically dependent static information into those .cdlTemplate

files, the program makeGridKeyTables to create the depictable and data keys, and the program processStyleInfo to generate style information.

This script has the capability to utilize non-default versions of the files dataFieldTable.txt, dataLevelTypeTable.txt, gridPlaneTable.txt virtualFieldTable.txt, and the \*.rules files. How these files are used to manage gridded data and the Volume Browser is a complex enough subject that it is treated separately in the documents [gridTables.html](#) and [styleRules.html](#).

#### **4.10) updateRadarFiles.ksh**

The script updateRadarFiles.ksh is controlled by the 'radar' task option. This script is used to create the data, depictable, product button and multi-load keys associated with nexrad radar data, as well as the needed menu structures. This is the only subordinate script written in the korn shell, and is probably the most complex, as it has six subordinate scripts of its own.

There are four files that control which radars are ingested and displayed and on which scales, all of which will have a default version generated if an override version is not supplied. radarsInUse.txt and radarsOnMenu.txt control, respectively, which radars are ingested and which radars appear on the menu. If needed, the default versions of these are constructed using radar metadata in the shapefile nationalData/fsl-w88d plus some distance tests in a utility script called radarUtil.csh. The files mosaicScales.txt and mosaicInfo.txt control which on-the-fly mosaics are available on which scales. The default versions of these files will set up a single mosaic available on scale 4 (usually the state scale) containing the nine nearest radars on the menu.

updateRadarFiles.ksh calls genRadarDataKeys.ksh to make data keys, genRadarDepictKeys.ksh to make depict keys, and makeRadarSupps.csh to generate depictor files for all of the radars known to the ingest. It also calls genRadarDataMenus.ksh to make menus, genRadarProdButtonInfo.ksh to make product buttons, and genRadarMultiLoadKeys.ksh to make multi-load entries for each radar on the menu. Finally, it calls doMosaicProcessing.ksh to create depict keys, product buttons, and menus for selected products that are displayed as on-the-fly mosaics.

updateRadarFiles.ksh and its subordinate scripts make use of several utility programs; shp2bcd, keyMunge, pasteUtil, and maksuparg. See [radarLocalization.html](#) and [radarMosaics.html](#) for more information about how radar localization works.

#### **4.11) makeMapFiles.csh**

The script makeMapFiles.csh is controlled by the 'maps' task option. The primary function of this script is to make files that the workstation can read to draw vector map backgrounds. It does this in three ways: first, it uses the program shp2bcd to convert shape files into either .bcd files (binary cartographic data files) or .bcx files (extended binary cartographic data files), which are the two formats the workstation can read to draw vector map backgrounds. Second, it uses

the program bcdProc to perform operations such as clipping and thinning on existing .bcd and .bcx files. Finally, it can move .bcd and .bcx files that are site specific control files into the localization data set.

#### **4.12) makeWWAtables.csh**

The script makeWWAtables.csh is controlled by the 'wwa' task option. The purpose of the makeWWAtables.csh script is to construct the geographic entity lookup tables (GELTs) that are needed by the WarnGen program, as well as the template files that are used to describe the specific format of each type of product that can be generated by WarnGen.

This script manipulates two types of files to make the files that serve as templates for the individual types of wwa products. Files in nationalData/ that have .preTemplate extensions are used to create files that have .template extensions in the localization data set. These files describe how to make generic blocks of text that can be used by many different wwa products, such as county, city, and zone lists, or UGC code headers. These files are brought into the specific templates for wwa products through an include mechanism. Files in nationalData/ that have .preWWA extensions are used to create files that have wwaProd extensions in the localization data set, and these files are the specific templates for WarnGen products. The manner in which both the list templates and the product specific templates are created is controlled by several different directives (see [Section 6.1.3](#)). For a description of how wwa product templates are interpreted by the WarnGen program, see [TextTemplate.html](#).

This script also uses the utility program newGELTmaker (see [newGELTmaker.doc.html](#)) to create geographic entity lookup tables (GELTs). These tables form the basis for how the WarnGen program is able to create descriptions of the area covered by a watch, warning, or advisory. newGELTmaker uses binary cartographic data files, shape files, and other files with ASCII lists of locations and identifiers (such as the CitiesInfo.txt file) as the raw material from which to construct geographic entity lookup tables. Currently, makeWWAtables.csh tries to construct eight different GELTs: a table for the county warning area of the localization, a wwa county table, a regional county table, a wwa zones table, a regional zones table, a warning cities table, a watch cities table, a wwa marine zones table, and a regional marine zones table. The files that control how these tables are made are called gelt script files. The default gelt script files live in nationalData/ and have file names that look like \*\_gsf.txt. Any override file named \*\_auto\_gsf.txt will result in the localization trying to automatically use that file as a GELT script file to create a GELT.

#### **4.13) makeStationFiles.csh**

The script makeStationFiles.csh is controlled by the 'station' task option. The purpose of the makeStationFiles.csh script is to construct station lists that contain progressive disclosure information in them. There are two types of station lists created here, station plot info files (.spi files), which are meant to be used to control the progressive disclosure for plots or

stochastically distributed hydrometeorological data, and location plot info files (.lpi files) which are just meant to be used for constructing map backgrounds with location information.

The source data for these are so called goodness files, which are files with a .goodness extension and have two possible formats, compatible with either creating .spi or .lpi files. It is also possible to create .lpi files from the .id file that is a component of a GELT or from the data in the file \$FXA\_HOME/data/CitiesInfo.txt file. The goodness files are so named because they contain a 'goodness factor' for each location, which is an arbitrary integer that says how desirable it is to see a given station at a low zoom level; the higher the number the more desirable it is to see the station at a low zoom. The program va\_driver is used by makeStationFiles.csh to convert these goodness factors into progressive disclosure parameters. The progressive disclosure parameter used in WFO-Advanced is the distance in kilometers to the nearest other station which is at least as likely to be seen at a given zoom.

makeStationFiles.csh uses the program shp2bcd to construct goodness files from shape files, the program GELTtest to restrict station lists to only those in a given geographic entity (such as a county warning area), and bcdProc to restrict station lists to only those within the area of a depictor file (which describes a display scale, for instance).

Station plot info (.spi), location plot info (.lpi), .goodness files, and CitiesInfo.txt are all plain ASCII, and so can be managed manually. The preferred method is to change the .goodness files and then rerun makeStationFiles.csh, rather than to edit .spi or .lpi files directly.

Finally, makeStationFiles.csh uses the utility program reformatTest to create any predefined static plan view plot data sets. For those instances where there are corresponding \*.cdl and \*.dat files, and there is a data key for an adaptive plan view plotting data set with a file name but no directory, it will attempt to generate a blank \*.nc file using ncgen on the \*.cdl file and then use the contents of the \*.dat file and the program reformatTest to initialize the file with data. See [staticProgDisc.html](#) for more information about how static progressive disclosure files are handled.

#### **4.14) makeDirectories.csh**

The script makeDirectories.csh is controlled by the non-default 'dirs' task option. The main purpose of the makeDirectories.csh script is to create the data directory for every entry in the dataInfo.txt table, including all of its include files. It also creates the directories for gridded data files and moves the \*.cdlTemplate files for each gridded data source into those directories and names them 'template', and creates the template files for plan view point data sets stored in netCDF.

#### **4.15) createAuxFiles.csh**

The script createAuxFiles.csh is controlled by the non-default 'auxFiles' task option. The main purpose of the createAuxFiles.csh is to create directories and files on \$FXA\_DATA that are not

standard data directories. Currently, this script sets up default RPS lists, creates the correct acquisition patterns, creates some files needed by the thunderstorm application, and copies some localization files to the ldad server.

#### **4.16) makePurgeTables.csh**

The script makePurgeTables.csh is controlled by the non-default 'purge' task option. The main purpose of the makePurgeTables.csh is to take the various files that describe how to purge grids, radar, satellite, and other keys and move them all into the localization data sets directory.

#### **4.17) fxatextTriggerConfig.sh**

The script fxatextTriggerConfig.sh is controlled by the non-default 'trigger' task option. The purpose of this script is to create text triggers, which can cause an action to occur upon arrival of a given text product.

#### **4.18) LAPS localization**

The Local Analysis and Prediction System (LAPS) is a high resolution hourly surface and 3D analysis that normally runs on the PX. LAPS is treated more or less as COTS by the WFO Advanced system, but it still needs some localization-generated geographic information in order to be properly configured. Invoking the non-default 'laps' task option will result in LAPS being configured to run over the local area associated with the ingest localization.

#### **4.19) MSAS localization**

The MAPS Surface Assimilation System (MSAS) produces an hourly surface analysis that runs on the PX, and is also used to quality control LDAD mesonet observations. MSAS needs some localization-generated information in order to be properly configured. Invoking the non-default 'msas' task option will result in MSAS being configured to run over the domain associated with the ingest localization.

The default MSAS domain for CONUS sites is a 60-km grid covering CONUS and neighboring areas of Canada and Mexico. For Alaska sites, the default is a 30-km grid covering Alaska, and for Puerto Rico, a 30-km grid covering Puerto Rico and nearby areas of the Caribbean is used. At this time, MSAS is not enabled for sites in the Pacific Region.

Each site can choose to modify the location, size, and resolution of its local MSAS domain. Changes in domain size are linked to changes in resolution in such a way as to minimize AWIPS impacts and guarantee that overall MSAS computational demands remain the same. For example, forecast offices could choose a 15-km, regional-scale domain, or a 60-km CONUS domain, but not a 15-km CONUS domain.

Each site can also specify the background grid utilized in the MSAS analysis. The default background over CONUS remains a linear combination of persistence and Eta-211, but forecast offices will also be allowed to specify that the AVN-213 be incorporated in the background grid, either in combination with persistence, or alone as a pure-model background. Off CONUS, the default background is a pure-model grid, using Eta-207 in Alaska and AVN-213 in Puerto Rico. Alaska sites can also choose AVN-213 to be used in the background grid.

Each site can also choose to redefine the "MSAS MSL Pressure" and "3hr Pressure Change" variables from builds 5.2.1 and earlier.

Pressure Reduction Level can range from 0 to 2000 meters. The MSAS pressure reduction algorithm will be used to produce a pressure analysis at that level. The default is 0, for sea level, and will match the original "MSAS MSL Pressure". Notice that if AVN 213 is used, the only pressure reduction level available is sea level (0 meters).

Pressure Change Interval can range from 1 to 6 hours. The default is 3, which will match the original "3hr Pressure Change".

For instructions on how to modify the default settings, see the `msas_sysdef.txt` file in `nationalData` (CONUS) or `localData/III-msas_sysdef.txt` for Alaska and Puerto Rico sites. For more information on MSAS localization, e.g., how and why to select different options, see <http://www-sdd.fsl.noaa.gov/MSAS/localization.html>

## 5. Files in the national data set

This section will cover all of the general categories of files that can be found in the national data set. The location, format and function of each type of file will be described. Whether and how its function can be overridden or augmented by override files will also be mentioned. For a more detailed itemized list of files in the national data set, see [fileChanges.html](#).

Site specific control files and realization files will be referred to with the prefixes 'LLL-' and 'RRR-' as before, where LLL is an arbitrary localization identifier and RRR is an arbitrary realization identifier (again not necessarily three characters). Files in the national data set that exist in `$FXA_HOME/data` will be referred to with the prefix 'data/' and ones that exist in `localization/nationalData` will be referred to with the prefix 'nationalData/'. Occasionally, a file will be referred to with the prefix 'III-', which is similar to the 'LLL-' prefix except that it refers to the identifier of the localization being used on the data ingest machine. All other first references to file names are assumed to be files in the final localization data set.

One concept that is very important to this section is file override. Here is an idealized example of how file override works. Suppose there is some file needed by the localization called `locFile.dat`. Very commonly, the existence of the following files will be checked: `data/locFile.dat`, `nationalData/locFile.dat`, `RRR--locFile.dat`, and `LLL-locFile.dat`. (It is also possible to provide override files from a directory pointed to by `$FXA_CUSTOM_FILES`, but the

focus here will be on files that exist in source code control. See [Section 9](#) for more on this.) The last of these files that actually exists is what is used by the localization. An additional understanding of how file override works can be gained by reading the header documentation of the `getPath.csh` and `fileGrab.csh` scripts.

In what will be referred to as functional override, the file is merely located using the `getPath.csh` script, in which case the file will be used as input data to create some other file or control some aspect of the localization process. Sometimes, functional override will be implemented by copying the files into the localization data set and then removing them once they have been used. In what will be referred to as copy override, the file will actually be moved using the `fileGrab.csh` script to create the file `locFile.dat` in the localization data set. In what will be referred to as replacement override, the override file is moved into the localization data set only when the RRR-- or LLL- version is present, and the workstation will rely on the `InfoFileServer` class to find it directly in the national data set otherwise. The `fileGrab.csh` script can also operate in append mode, which means it would try to concatenate each of these files to `locFile.dat` in the localization data set with the `>>` redirect operator. That way, the national data set version will be at the beginning of the file in the localization data set and the local override version would be at the end. This final override method is referred to as append override.

When a file is subject to copy override or append override, a version of that file is always created in the localization data set. When a file is subject to functional override, no version of that file is ever permanently created in the localization data set. When a file is subject to replacement override, a version of that file is created in the localization data set only when the selected version is a local or realization file. This point is very important; it is possible for the same file to be subject to different types of override from different sources. For example, a file might be subject to copy override if an RRR-- version was found, but then subject to append override if an LLL- version was found.

File override selection always operates on a file by file basis. The `fileGrab.csh` script can operate on groups of files, so it is possible that one use of the `fileGrab.csh` script would result in files originating from several different source directories all ending up in the localization data set.

For every file mentioned in the national data set catalog, what type of file override it is subject to will be mentioned. Some files are not subject to any override, and this will be stated as well. When a file is 'renamed', this means that it is copied to the localization data set and given a new name; files in the national data set are never removed or changed by the localization process. When a file is renamed, it is usually changed by `sed` or some other program.

### **5.1) Scale table**

The file `nationalData/scaleInfo.txt` contains a list of all of the scales for the localization, along with names, depictor files and default maps for each scale. `scaleInfo.txt` is subject to copy override, with the caveat that comments are stripped as it is moved.

## 5.2) Process config files

Process config files are files that control how a process in the workstation or data ingest behaves. The file `data/fxa.config` is the main process config file. It actually just contains five include files, `'ws.config'`, `'scales.config'`, `'ipc.config'`, `'text.config'`, and `'wwa.config'`. `ws.config` is where such things as default looping parameters and zoom levels are defined. `scales.config` is where the startup scale index for each IGC is set. `ipc.config` is where the interprocess communication targets are set. The file `text.config` is generated from the file `textConfig.template` and for now just holds the AFOS CCC. The file `wwa.config` is generated from `wwaConfig.template` and contains the list of active WarnGen product titles and file names, as well as a list of the alternate localizations with which it is allowable to restart WarnGen. `fxa.config`, `ws.config`, `scales.config`, and `ipc.config` are subject to replacement override. `wwaConfig.template` and `textConfig.template` are subject to functional override.

## 5.3) Main and manual data and depict key files

The files `data/dataInfo.txt` and `data/depictInfo.txt` are not subject to file override. These are the top level files that the depictable key and data key table modules read. These files actually only contain `#include` statements that bring in files for manually maintained keys, satellite keys, radar keys, and grid keys. The files in `localization/nationalData` called `dataInfo.manual` and `depictInfo.manual` contain the manually defined data and depictable keys, as well as `#include` statements that bring in text keys and keys just defined for the localization. These files are subject to replacement override, and have extensive header documentation in them that is meant to provide information needed to maintain the file.

## 5.4) Text database and depictable localization

The files in the directory `localization/nationalData` called `versions_lookup_table.template`, `national_category_table.template`, and `ispan_table.template`, are subject to functional override. The active versions of these files are renamed to `versions_lookup_table.dat`, `national_category_table.dat` and `ispan_table.dat`. During the move, `sed` commands replace occurrences of the string `@@@` with the AFOS CCC. `versions_lookup_table.dat` also has purge parameters for METAR text products added based on distance from the localization center. The file `nationalData/ccclatLon.txt`, which contains a location for each AFOS CCC, is used to determine these distances. The files in `nationalData/` called `textDataKeys.template` and `textDepictKeys.template` are the raw data that specify how data and depictable keys for text based displays are generated, and are both subject to functional override. When `makeTextKeys.csh` and `textUtil.csh` create depictable key and data key entries for text depictables, the file `nationalData/afosMasterPIL.txt` is used as a final sanity check for whether any given text product can actually be used in a text depictable, and the file `nationalData/stateMatch.dat` is used to get a state ID from an XXX where needed. `afosMasterPIL.txt` also has `@@@` as stand in for the local CCC. While `ccclatLon.txt` is not subject to any override, `afosMasterPIL.txt` and `stateMatch.dat` are subject to functional override.

## 5.5) Satellite data and depictable keys

The files in localization/nationalData called eastSatDataInfo.template, eastSatDepictInfo.template, westSatDataInfo.template, and westSatDepictInfo.template are used to create the files which comprise the data and depictable key entries for satellite data. There are east and west versions because it is possible for any given localization to be served by either the east or west satellite. The central position of the localization being used by the data server is always written into the file 'IngestCenter.dat' by the makeScales.csh script when it is called by makeClipSupps.csh. If the longitude in IngestCenter.dat is east of 100W, then the east files are used, otherwise the west files are used. This default behavior can be changed with the SATEW directive. None of these files are subject to override. Assuming xxxx is either east or west, as appropriate, then xxxxSatDataInfo.template and xxxxSatDepictInfo.template are renamed to satDataInfo.txt and satDepictInfo.txt. During the move, a sed operation replaces '@@@' strings with the localization ID of the data server for satDataInfo.txt and both files have their comments stripped. satDataInfo.txt and satDepictInfo.txt are both pulled into the main data and depict key files through #include statements.

Although this is not the usual procedure, it is also possible to obtain versions of satDepictKeys.txt and satDataKeys.txt directly through copy override.

## 5.6) Product button file

The file nationalData/productButtonInfo.txt contains a table of all of the default product buttons, as well as a #include statement that can bring in locally defined product buttons. The productButtonInfo.txt file is subject to replacement override.

## 5.7) Menu files

This section deals with menu files in general. Radar menus will be discussed at greater length in a later section. There are currently fifteen menu files that exist in the directory localization/nationalData/ and are under source code control: dataMenus.txt, backgroundMenus.txt, aircraftMenus.txt, aviationDataMenus.txt, commonLdadMenus.txt, mosaicDataMenus.template, otherUaMenus.txt, radarDataMenus.template, scanDataMenus.template, tdlAnalysisMenus.txt, tdlRadarBackgroundMenus.template, tdlRadarDataMenus.template, tdlSurfaceMenus.txt, tdlToolMenus.txt, and tdwrDataMenus.template. Additionally, there are seven menu files that exist by default in localization/nationalData/ which are downloaded from the noaa1 server: raobMenus.txt, redbookHazardMenus.txt, redbookNCOMenus.txt, redbookHPCMenus.txt, redbookCPCMenus.txt, redbookMarineMenus.txt, and redbookUpperAirMenus.txt. These files are subject to replacement override. The default version of dataMenus.txt contains the main default set of menu entries for the workstation, and has many #include statements in it. The include statements for raobMenus.txt, raobLocalMenus.txt, and radarDataMenus.txt, plus those listed above all invoke default functionality; all others are activated by the presence of an override file. The default set of menus for RAOBs is in raobMenus.txt, and the file

raobLocalMenus.txt is constructed from it using raobUtil.csh. raobLocalMenus.txt contains a local RAOB menu with about 5 to 10 sites close to the center of the localization, and is used to place the 2 to 4 closest RAOBs directly on the Upper Air menu. The NexRad radar menus are brought in with radarDataMenus.txt, and the map background menus are brought in with backgroundMenus.txt. The aircraftMenus.txt file brings in entries for PIREP and MDCRS displays, commonLdadMenus.txt brings in entries for LDAD displays, the ones starting with 'redbook' bring in entries for redbook graphics, and the ones starting with 'tdl' bring in entries for functionality developed by MDL.

## **5.8) Shape files**

A shape file is actually made up of three components; a '.dbf' file, a '.shp' file and a '.shx' file. All of the shape files currently used in the localization process currently are found in the directory localization/nationalData, and are not subject to override. Shape files are the primary repository for the national configuration data set provided by NWS headquarters. At present there are five different shape files available, and they all contain data sets for the entire conterminous U.S. Most important to the localization process are usa\_cwa, uscounty, c11-zone, and fsl-w88d, which contain information about county warning areas, counties, forecast zones, and Nexrad radars, respectively. One can also find usa\_lake, marine\_zones, timezones, and basins, us\_inter, which contain information about lakes, marine forecast zones, time zones, river basins, and interstate highways, respectively.

Because of their size, the files c11-zone.shp, usa\_cwa.shp, and uscounty.shp are kept in the source tree in compressed form. Thus, when new versions of these files are made available from NWS headquarters, they must be compressed before being checked into the source tree. The software that reads them will automatically uncompress them when required. In an environment without a source tree, it is perfectly OK to just replace these files directly in localization/nationalData without worrying about compressing them.

Shape files control a number of things. They are the primary source of data for creating map backgrounds. The county warning area information in the shape file usa\_cwa provides the basis for defining the scales for all of the different localizations. Shape files are also the main source of data for creating geographic entity lookup tables, which form the basis for how the WarnGen program is able to create descriptions of the area covered by a watch, warning, or advisory.

## **5.9) Vector map background files**

There are two AWIPS-specific file types that the workstation reads directly to create vector map backgrounds; binary cartographic data files (.bcd files) and extended binary cartographic data files (.bcx files). The .bcd file type is used for drawing lines only, .bcx files allow annotated lines to be drawn. For example, a .bcd file is used to draw county boundaries, a .bcx file is used to draw interstates. For county names, because one would label the center of the county instead of the border, a separate file containing a location list is used. AWIPS can also read shape files directly: see [shapeFileDisplay.html](#) for more information about this.

Currently, all .bcx files and many .bcd files are created using the utility program shp2bcd with shape files as input. The utility program bcdProc can also perform many other operations on .bcd and bcx files. The .bcd files that do not come from shape files all exist in the \$FXA\_HOME/data directory. These files are all subject to replacement override, and are used to describe data sets that we do not yet or will not have available from shape files. Some examples of this are continental and other international boundaries, snowfall contours for an orographic snow model covering the mountains of Colorado, some highway and river data for Colorado, and ARTCC boundaries.

### **5.10) Topography files**

In nationalData there are five files containing raw topography data; usTopo.dat.gz, worldTopo.dat.gz, akTopo.dat.gz, pacTopo.dat.gz, and caribTopo.dat.gz. These are used as the raw data to create topography grids and images specific to the display scales. Because of their size, these files are kept in the source tree in compressed form; the program that maps the data to specific scales can decompress them on the fly. None of these files are subject to any override behavior. These files are binary flat files that are grids of two byte integers, the values of which are elevation in meters. The file usTopo.dat is a one kilometer grid of data covering the conterminous U.S., and worldTopo.dat is 2.5 minute data covering the whole world. The file akTopo.dat is a one kilometer grid of data covering Alaska, and pacTopo.dat and caribTopo.dat are 30 second data covering the Pacific and Carribean, respectively. The utility program test\_grhi\_remap is used to remap this data and to convert it into the format that the workstation needs. Normally, three netCDF files are created that hold high resolution image topography for the state, CONUS, and Northern Hemisphere scales. ASCII flat files are created for LAPS topography and clipped mesoEta topography.

### **5.11) Gridded data and Volume Browser tables**

There are five tables which control the overall characteristics of gridded data and the Volume Browser: gridSourceTable.txt, dataLevelTypeTable.txt, gridPlaneTable.txt, dataFieldTable.txt, and virtualFieldTable.txt. The file gridSourceTable.txt is managed in the makeGridSourceTable.csh script, the rest being managed in the updateGridFiles.csh script. A version of all these files except for gridSourceTable.txt exists in the localization/nationalData directory; gridSourceTable.txt is built primarily from nationalData/gridSourceTable.template. All these file are subject to replacement override from realizations and append override from other override files, except for gridSourceTable.txt. This file is primarily overridden from localGridSourceTable.txt, by a mechanism that is a hybrid of include and append override. The localization will also append nationalData/tdlGridPlaneTable.txt, nationalData/tdlVirtualFieldTable.txt, nationalData/tdlDataFieldTable.txt, nationalData/tdlDataLevelTypeTable.txt, nationalData/tdlSourceTable.template to the respective files mentioned above.

If a lll-gridSourceTable.txt file is available for the data server localization, then it will be used as the core of the source table, otherwise, the file nationalData/gridSourceTable.template will

provide the core. The localization will also append the file `nationalData/tdlSourceTable.template` to the core and attempt to append `localGridSourceTable.txt` to the core. Both `gridSourceTable.template` and `tdlSourceTable.template` are subject to functional override. To create the `gridSourceTable.txt` file, a sed command is invoked on the core to tag the depictor files for LAPS and mesoEta with the data server localization identifier. `gridSourceTable.template` also has all gridded data sources marked as inactive. The file `nationalData/activeSources.txt` contains a list of unique IDs for those gridded data sources to activate. It is subject to copy override for realizations and append override for localizations. The file `inactiveSources.txt`, if present as a site specific control file, can be used to turn off sources.

The structure of the netCDF files that contain the gridded data for each source are defined in `.cdl` files. The `.cdl` files for all of the default data sources exist in the `$FXA_HOME/data` directory, and are subject to replacement override.

Briefly, it is the job of the utility program `makeGridKeyTables` to use these five tables and the `.cdl` files for the individual gridded sources to create the files `gridDataKeys.txt`, and `gridDepictKeys.txt`. It is the job of the utility program `testGridKeyServer` to create the depictors and Volume Browser menus for the predefined latitude and longitude cross section baselines, as well as the Volume Browser menus for selecting the data source. To see a detailed write-up of how this works, please see [gridTables.html](#), [makeGridKeyTables.doc.html](#), and [testGridKeyServer.doc.html](#).

## 5.12) Display style files

Style information is metadata that controls the look and feel of how data is displayed, such as contour intervals, or how to label the color bar. There are eleven primary files in the directory `localization/nationalData/` that are used to control style information: `contourStyle.rules`, `gridImageStyle.rules`, `iconStyle.rules`, `arrowStyle.rules`, `barbStyle.rules`, `graphStyle.rules`, `streamlineStyle.rules`, `imageStyle.txt`, `radarImageStyleInfo.template`, `radarGenericImageStyle.txt`, and `tdwrlImageStyleInfo.template`. The files `nationalData/tdlContourStyle.rules` and `nationalData/tdlGridImageStyle.rules` are appended to their respectively-named files before the rules files are processed. The `*.rules` files are subject to replacement override for realizations and to append override for localizations, and are used by the `processStyleInfo` utility program in `updateGridFiles.csh` to create `contourStyle.txt`, `gridImageStyle.txt`, `iconStyle.txt`, `arrowStyle.txt`, `barbStyle.txt`, `graphStyle.txt`, and `streamlineStyle.txt` which contain all of the style information for products from the Volume Browser. For more details on how these work, see [styleRules.html](#).

The file `nationalData/imageStyle.txt` is managed in `assembleTables.csh` and is also subject to replacement override. It contains manually maintained style information for all images except those from gridded data, nexrad data, and topography data. Style information for gridded data images is in `gridImageStyle.txt` and style information for nexrad images comes from `radarImageStyleInfo.template`, `radarGenericImageStyle.txt`, or from within the data itself. See

the next section for more information on radar style information. Style information for topography images is written to the file `topoImageStyle.txt` and is generated by the same `test_grhi_remap` utility program that generates the images. There is a `#include` statement in `imageStyle.txt` that is used to bring in the `topoImageStyle.txt` file. There is also a `#include` for the file `localImageStyle.txt`; if a site wants to modify or add image style entries for displays other than radar or grids locally, it should be done here rather than changing these other files. There is also extensive header documentation in `imageStyle.txt` that is meant to provide information needed to maintain the file.

### 5.13) Radar-related files

The file `radarInfoMaster.txt` is a master list of all nexrad radars, with their locations, identifiers, and immutable indices, and is subject to non-standard override. If an `III-radarInfoMaster.txt` or `RRR--radarInfoMaster.txt` exists, then that file will be used as is. Otherwise, the `shp2bcd` utility program is used to generate it from the shape file `nationalData/fsl-w88d`, which is not subject to override. The files `radarsInUse.txt` and `radarsOnMenu.txt` control, respectively, which radars the ingest knows about and which radars appear on the menu, and these files are both subject to a similar non-standard override functionality. If the files `III-radarsInUse.txt` or `LLL-radarsOnMenu.txt` are available, then they will be used as is, otherwise they will be generated from the `fsl-w88d` shape file, using distance tests to make an estimate of which radars are appropriate for ingest and display. The files `mosaicScales.txt` and `mosaicInfo.txt` are subject to a very similar non-standard override; they control the availability of on-the-fly mosaics.

There are several `.template` files in `nationalData/` that control what key, button, and menu entries for each nexrad radar look like. In these files the string `'@@@@'` is a stand in for an arbitrary radar ID, and a single `@` is a stand in for a key list. All of the keys in these files are generic keys in the range 10000-65535. Except for areal composite product keys, all of these keys are converted into radar specific keys based on the arbitrary immutable index for each radar. `radarDataKeys.template` contains generic entries for nexrad data keys, `radarDepictKeys.template` contains generic entries for nexrad depict keys, and `radarProductButtonInfo.template` contains generic entries for nexrad product buttons. The file `radarMultiLoadInfo.template` contains generic entries for multi-loads, which is how the workstation handles such things as reflectivity/velocity combo and four-panel displays. The file `radarDataMenus.template` serves as a template for creating the menu entries for one nexrad radar. A similar set of files exists for TDWR radars: `tdwrDataKeys.template`, `tdwrDepictKeys.template`, `tdwrProductButtonInfo.template`, `tdwrMultiLoadInfo.template`, and `tdwrDataMenus.template`. The 'Radar' menu is where the user accesses mosaics and dial radars. The files `mosaicDataMenus.template`, `mosaicDepictKeys.template`, and `mosaicProductButtons.template` contain generic entries of menus, depict keys, and product buttons for areal composite products. All of the files mentioned in this paragraph are subject to append override.

Style entries for radar data are controlled in three files. Files `radarImageStyleInfo.template` and `tdwrImageStyleInfo.template` work just like the files mentioned above and are used to create

entries in the output files radarImageStyleInfo.txt and tdwrImageStyleInfo.txt, respectively, which are radar-specific keys that control how individual radar images are displayed. The file radarGenericImageStyle.txt is not processed to produce radar specific keys; it stays as is. This file contains the display style for mosaics and the style which describes the image count to data value mapping for 8-bit products.

For more information on managing localization for radar displays, please see [radarLocalization.html](#) and [radarMosaics.html](#).

#### **5.14) Files related to WarnGen**

The file nationalData/wwaDefaults.txt contains some default values of some directives for generating WarnGen product templates. See [Section 6.1.3](#) for more details on how these directives work.

The files in nationalData/ with names like wwa\_blahblah\_blah.preWWA are source data for the default set of WarnGen product templates. Each file like this gets converted by the makeWWAtables.csh script into a file that look like wwa\_blahblah\_blah.wwaProd, and these files are the actual WarnGen product templates. The \*.preWWA files are subject to functional override. In the conversion process, sed commands are used to substitute the value of several directives in these files (see [Section 6.1](#)). For every WarnGen product template, entries are made in the wwa.config file, which are the name of the file and the title of the product; this registers the file and its description to the WarnGen program. The title of the product must be present in a commented line (// style) in the .preWWA file with the title in quotes. WarnGen templates are sorted on the WarnGen menu based on the title; any text in the title that occurs before an optional vertical bar(|) is used only for sorting purposes and will not appear on the WarnGen menu. The function of the WarnGen product template files is described in [TextTemplate.html](#).

WarnGen product templates can access through #include statements one or more files with names like wwa\_blah\_blah.template. These files contain unified mechanism for generating UGC codes, or lists of items such as counties, cities, or zones. The source data for these files are files in nationalData/ with names like wwa\_blah\_blah.preTemplate. The makeWWAtables.csh script converts the \*.preTemplate files into the \*.template files. In the process, sed commands are used to substitute the value of several directives in these files (see [Section 6.1](#)). The \*.preTemplate files are not subject to any override functionality. The functionality that performs this conversion is somewhat complex; sometimes the wwaUtil.csh is used to put several different versions of a \*.preTemplate together into one \*.template file.

A related group of files is the nationalData/\*.abbrev files, of which there are currently three: areas.abbrev, county\_type.abbrev, and state.abbrev. These files are used by WarnGen to translate abbreviations into plain language, and are subject to copy override only from realization files. They refer to parts of states, how to describe sub-state political divisions in areas that don't have counties, and two-letter postal codes.

Files in nationalData/ that have file names like \*\_gsf.txt are referred to as GELT script files. These files are subject to functional override from all places. They are used by the program newGELTmaker to create Geographic Entity Lookup Tables. Also many of the files that the program newGELTmaker typically reads in are subject to various types of override, as well.

### **5.15) Files that control stochastic progressive disclosure**

The file data/MTR.goodness and files in localization/nationalData that have '.goodness' extensions are the most common of the files that control stochastic progressive disclosure. These files are ASCII station or location lists in which each line in the file refers to one station or location. The term station is meant to apply to a site at which some hydrometeorological data is available for plotting, whereas location refers to a site that is used merely for georeferencing (i.e. for drawing a map background). Both station and location lists contain a latitude, longitude, ASCII identifier, and an arbitrary desirability factor. These arbitrary desirability factors have acquired the name 'goodness values', hence the file naming. A goodness value is an arbitrary integer for which larger values mean that a given station or location is more likely to be viewable on the screen for a given zoom factor. The script makeStationFiles.csh hands these .goodness files to the program va\_driver to create either station plot information files (.spi files) or location plot information files (.lpi files). It is the job of va\_driver (see [va\\_driver.doc.html](#) for more information) to convert the arbitrary goodness values to progressive disclosure distances that the workstation can use to perform stochastic progressive disclosure. The file data/MTR.goodness differs from the other .goodness files in that it is updated by a program rather than maintained manually. All .goodness files are subject to functional override. The goodness files that are currently available in the national data set are MTR.goodness, 88D.goodness, BUOY.goodness, profiler.goodness, raob.goodness, twebRoutes.goodness, and twebStations.goodness. These contain information about METAR stations, Nexrad radars, stationary buoys, profiler, RAOBs, TWEB route markers, and TWEB route anchor points.

An additional source of information for controlling stochastic progressive disclosure is the file data/CitiesInfo.txt. This file is the main source of information for constructing the cities map background, as well as the WarnGen cities table and the warning cities map background. The format of this file is also documented in [va\\_driver.doc.html](#).

Another type of file that affects stochastic progressive disclosure is files that have a '.primary' extension. These are lists of station or location identifiers that are to be made visible at lower zoom factors regardless of the goodness value associated with them. These files are subject to reverse append override.

Two related files are data/anchors.txt and data/selsAnchors.txt. These are used for labeling cross section baselines and decoding SAW products respectively. These are not subject to any file override and (especially selsAnchors.txt) should normally be centrally maintained, but it is useful to know about these files in case either of these functions fails to work as expected.

### **5.16) Color table files.**

The source data for the default color tables delivered with the system is the file `data/colorMaps.mark`, and this file is subject to functional override. The file `colorMaps.nc` is the file that is read at run time to get default color table information; this is a netCDF file which is created by `assembleTables.csh` by doing an `ncgen` on `colorMaps.mark`. When custom color tables are created by using the color table editor, they are written to `data/customColorMaps.nc`. Doing an `ncdump` on this file after creating a custom color table is the best source of data for making permanent modifications to the default color tables. Such modifications are best placed in the file `LLL-localColorMaps.mark`, with table numbers changed to be in the 500-999 range. The script `assembleTables.csh` will use an `ncgen` command to create `localColorMaps.nc` from this file; the color tables in `localColorMaps.nc` will also become part of the default set.

### **5.17) Run time config files.**

The workstation reads the file `data/fxa.config` at start-up to define many things, such as allowable zoom factors and which scales each window starts with, to name a couple. Unlike the `mainConfig.txt` and `wwaConfig.txt` files, which just control the way localizations run, `fxa.config` can directly change the way the workstation and ingest software behave. The file `fxa.config` actually just contains a bunch of include statement for the files `ws.config`, `scales.config`, `ipc.config`, `text.config`, `wwa.config`, and `tdl.config`. These files control aspects of, respectively, display configuration, start up scales, interprocess communication, text data retrieval, WarnGen, and other miscellaneous configurables needed by non-FSL developers. The files `text.config` and `wwa.config` are actually generated by the localization from the files `nationalData/textConfig.template` and `nationalData/wwaConfig.template`, both of which are subject to functional override. The rest of the `*.config` files exist in their default runtime states in `nationalData/`. These are subject to copy override from realization files and append override from site specific control files. Append override works well for these because duplicate entries for the same configuration item will result in the last entry being used.

### **5.18) Files which control plan view plotting.**

There are two main types of files that exist specifically to control plan view plotting, design files and lookup table files. The default versions of both types of files live in `nationalData/`. Design files have file names that look like `*Design.txt`, and lookup table files have file names that look like `*_*.txt`. Both are subject to copy override from realization, site specific, and customization files. Briefly, a design file allows one to specify the layout of a single plan view plot display, and lookup table files are used during the display of plan view plots to do arbitrary data conversions. The adaptive plan view plotting capability, which is new to AWIPS for build 5, is complex enough that a separate document exists that describes it in detail. See [adaptivePlanViewPlotting.html](#).

### **5.19) Files that specify the MSAS domain, background, and user-definable variables.**

The following nationalData/ files are used to specify the MSAS domain, background, and user-definable variables: msas\_sysdef.txt, msasDepictKeys.txt, msasFieldConfig.txt, and msasProductButtons.txt. The msas\_sysdef.txt file is used to specify the domain and options. The other files are modified by internal MSAS programs, using information from msas\_sysdef.txt, and should not be edited or replaced by the user. For more information, see section 4.1.9 above, or go to the MSAS localization Web pages at: <http://www-sdd.fsl.noaa.gov/MSAS/localization.html>

## 6. Other site specific control files

In the previous section, which discussed the files in the national data set, site specific control files were mentioned, but only in the context of where they overrode the functionality of analogous files in the national data set. This section will focus mainly on files with no direct analog in the national data set.

This discussion will use the same file identification convention as the previous section. Specifically, the prefixes 'LLL-' and 'RRR--' will refer to site specific control files and realization files, the prefixes 'data/' and 'nationalData/' will refer to national files in the directories \$FXA\_HOME/data and localization/nationalData, and all other references to file names are assumed to be files in the final localization data set.

### 6.1) Localization config files

Localization config files are files that control how the localization scripts behave. They have the file name pattern LLL-blahblahConfig.txt and contain items called directives (also see [directives.htm](#)). A directive is a single line in a file that looks like

```
@@@DDD argument
```

Each directive begins with 3 at signs followed immediately by a directive type, which is not necessarily 3 characters, but is all upper case by convention. The 'argument' can be any arbitrary text; how it is used varies among different directives. There are currently two types of localization config files.

#### 6.1.1) LLL-mainConfig.txt

The main purpose of this file is to contain directives that can define additional localizations. There are two directives that can be used to define a localization. The first is the 'WFO' directive, which must have as its argument one of the county warning area identifiers in the usa\_cwa shape file in localization/nationalData. Executing the script cwalds.csh in localization/scripts will provide a list of these county warning area identifiers. The second is the 'CLONE' directive. The argument of the CLONE directive must be the identifier of some other valid localization which is not itself defined by a clone directive. A localization defined by the WFO directive will not inherit any site specific files from the localization with the same name as the county warning area identifier, whereas a localization defined by the CLONE will inherit any site specific files

from the localization it is cloned from. Any site specific files belonging to the new cloned localization will override any from the localization it is cloned from.

The final type of directive that can define a localization is the 'REALIZATION' directive. The REALIZATION directive has as its argument the name of the realization to associate this localization with (see chapter 8). For a complete list of usable directives, the reader is directed to [directives.html](#).

#### 6.1.2) LLL-cwa.asc

This file does not contain directives, but it is related to the WFO directive in LLL-mainConfig.txt. This file can be used to define a localization centered about some arbitrary point or area, not just one of the predefined county warning identifiers. This file contains any number of latitude-longitude points. Each point is on one line in the file, and is in ASCII format, space delimited. The localization ends up being centered around that point or group of points. It is not recommended to define a localization with both a LLL-cwa.asc file and either a WFO or CLONE directive in the LLL-mainConfig.txt file.

#### 6.1.3) LLL-wwaConfig.txt

This file contains directives that change how the WarnGen functions. A file in nationalData/ called wwaDefaults.txt contains default settings for some of these directives, but any instances of these directives in LLL-wwaConfig.txt will override those in wwaDefaults.txt. For a complete list of the directives that one can place in the wwaConfig.txt file, one is again directed to [directives.html](#).

### 6.2) Files referred to through #include

There are several cases where site specific control files do not override an existing file in the national data set, rather they are referred to by #include statements that already exist in files found in the national data set. These files do not have to be supplied; if they are not the software that ingests these files will just ignore #include statements that cannot be resolved. All files referred to here must originate as LLL- or RRR-- files to become part of the localization.

#### 6.2.1) Depict keys, data keys, and product buttons

The file nationalData/dataInfo.manual has a #include statement for bringing in the file localDataKeys.txt, which must come from the file LLL-localDataKeys.txt. This allows any arbitrary additional data keys to be added that are specific to the localization. Also, in nationalData/depictInfo.manual, there is an analogous #include statement that refers to localDepictKeys.txt. Finally, in the file nationalData/productButtonInfo.txt, there is a #include statement for the file localProductButtons.txt, which can bring in any arbitrary additional product buttons that are specific to the localization.

## 6.2.2) Menu files

Unlike depict keys, data keys, and product buttons, which are order insensitive, entries in the menus files are order sensitive. Thus, there are many different places in the default menus where site specific control files can be brought in by #include statement. The file nationalData/backgroundMenus.txt, which is the default map background menu, has a #include statement for the file otherBackgroundMenus.txt. In nationalData/dataMenus.txt, there are many #include statements designed to bring in site specific control files. The files otherToolMenus.txt and otherVolumeMenus.txt allows localization specific additions to the Tools and Volume menus. The file ldadMenus.txt is where non-default LDAD data plots are added to the surface menu. The file analysisMenus.txt allows localization specific surface analysis menus to be brought in, and otherSurfaceMenus.txt allows any other arbitrary additions to the end of the surface menu. Finally, otherUaMenus.txt and otherSatMenus.txt allow for arbitrary additions to the end of the upper air and satellite menus.

## 6.3) File override expansion

So far, when file override has been discussed, it has been talked about in terms of a localization or realization file directly replacing an existing file in the national data set. There are cases where site specific control files can add to a group of related files, rather than just replace existing files in that group. Usually, when this is done, it is not enough to just add the new file; something else must be changed so that the system can recognize the new file as something that should be used by the workstation.

### 6.3.1) Menu files

Any file in the localization specific files (LLL- files) or realization specific files (RRR- files) that has a file name pattern like '\*Menus.txt', '\*MenuHeader.txt', or '\*MenuFooter.txt' will be moved into the localization data set. However, to become a useful menu, one of the existing default menus must be overridden and a #include statement for the new menu file added.

### 6.3.2) Map background files

Any file in the localization specific files or realization specific files that has a file name pattern like '\*.bcd' or '\*.bcx' will be moved into the localization data set. A new file such as this cannot be used for drawing a map background until data key, depict key, product button, and menu entries have been defined. In this case, this would usually be handled using the files LLL-localDataKeys.txt, LLL-localDepictKeys.txt, LLL-localProductButtons.txt, and LLL-otherBackgroundMenus.txt, respectively.

### 6.3.3) Topography and cdl files for gridded data sources

Any file in the localization specific files or realization specific files that has a file name pattern like '\*.cdl' or '\*.topo' will be moved into the localization data set. These files are associated with

gridded data sources. The .cdl files are used to define the exact variables and levels stored for a given gridded data source, and the .topo files contain the surface topography used for a given gridded data source. Before new files like these can be used by the system, one must also add a new gridded data source to the system that specifically refers to these new .cdl and/or .topo files. This can be done by functionally overriding the file in nationalData/ called gridSourceTable.template, or by providing a file called III-gridSourceTable.txt, which will be used as a direct replacement for the grid source table.

#### 6.3.4) Product templates for WarnGen

Any file in the localization specific files or realization specific files that has a file name pattern like 'wwa\_\*.preWWA' will be moved into the localization data set. These files are used to create analogous files with names like 'wwa\_\*.wwaProd' that serve as WarnGen product templates. In order for a new file like this to be used by the WarnGen program, one must make sure it has a title line, which is a commented out line (// style) with the title in quotes immediately after the comment.

#### 6.3.5) GELT script files

GELT script files are files that contain instructions that are used by the program newGELTmaker to create Geographic Entity Lookup Tables, which are used by WarnGen to identify which counties, cities, etc., fall within a warned area. GELT script files have file names that look like \*\_gsf.txt, and the defaults live in nationalData/. Any file in the site specific, realization, or customization files that has a file name pattern like this will be used to try to create a GELT. These override files can be used to change how a default GELT is created or to generate additional GELTs.

#### 6.3.6) Station lists

Any file in the localization specific files or realization specific files that has a file name pattern like '\*.goodness' and '\*.primary' will be used to create either station plot information (.spi) files or location plot information (.lpi) files in the localization data set, depending on the format of the .goodness file. A .primary file only modifies how a .goodness file is interpreted, so a .primary file by itself is meaningless (unless it refers to a .goodness file already in the national data set). In order for the resulting .lpi or .spi files to be used to create point based map backgrounds, entries must be made for data keys, depict keys, product buttons, and menus. In this case, this would usually be handled using the files LLL-localDataKeys.txt, LLL-localDepictKeys.txt, LLL-localProductButtons.txt, and LLL-otherBackgroundMenus.txt, respectively.

### 6.4) Files used to activate and deactivate data sets

The files activeSources.txt and inactiveSources.txt are subject to copy override from realization files and to append override from site specific files. By default, no gridded data sources are

active. They become active in the default system by virtue of the fact that a national data set version of activeSources.txt exists. After applying file override to build versions of these files, all sources mentioned in activeSources.txt but not mentioned in inactiveSources.txt will be activated.

The files removeMenuItems.txt and preserveMenuItems.txt are subject to copy override from realization files and to append override from site specific files. They allow a localization to have certain menu items removed from the user interface without having to edit or override files in the default menus. After applying file override to build versions of these files, menu items listed in removeMenuItems.txt but not in preserveMenuItems.txt will be removed from the user interface. Each menu item listed contains an object to delete from menus plus an optional leading comma delimited file name. If the file name exists, then that will be the only that file name will be searched for items to delete. The object to delete can be a button number, an appButton name, an include file name, or a submenu name. If a submenu name, use a '.' for spaces or metacharacters. In the case of a submenu name, it will delete the whole submenu, accounting for nesting, but will not do the right thing for files where the submenu statement is in a different file from the endSubmenu statement.

## **7. Localization programs**

This chapter will give short descriptions of each of the utility programs used by the localization process. For detailed user documentation for these programs, one should see the corresponding individual documentation files.

### **7.1) fileMover**

The utility program fileMover (see [fileMover.doc.html](#)) is new for 5.1.1. Just about every time a file is brought into the localization data set, the utility program fileMover is used, as opposed to a unix cp, cat, or mv command. Among other things, it is used to reliably strip always end in a newline. In fileMover is implemented a feature that allows one to change the type of override a file is subject to. If a file is normally subject to replace override, one can change that to append override by making the first line in the file literally '#append'. The converse can be done with a first line of '#replace'.

### **7.2) Programs that create depicter files**

Depicter files are the files within the WFO advanced system that describe a frame of reference. The most common of these are geographic depicter files, which have .sup extensions. These are most often generated by the program maksuparg (see [maksuparg.doc.html](#)). Those geographic depicter files that represent some portion of a data grid are sometimes generated by the program maksubgrid (see [maksubgrid.doc.html](#)). Depicter files that describe a thermodynamic diagram have a .thermo extension and are generated by the program makthermo (see [makthermo.doc.html](#)). Depicter files that describe a cross section have a .xsect extension and are generated by the program makxsect (see [makxsect.doc.html](#)).

### **7.3) Programs that manipulate key files and menu files**

The program keyMunge (see [keyMunge.doc.html](#)) is used to take generic radar keys and make them radar specific. The program pasteUtil (see [pasteUtil.doc.html](#)) functions much like the unix paste utility, and is used to create both key tables and station lists. The program rangeAzimuth (see [rangeAzimuth.doc.html](#)) can do distance and bearing calculations between two points and is used to generate menu entries based on the distance from the site.

Finally, many of the key tables in WFO advanced are actually put together at run time through the use of an include mechanism. The program textBufferTest (see [textBufferTest.doc.html](#)) can be used to locate and output to standard output the entire contents of such a table.

### **7.4) Programs that manipulate cartographic data sets**

The program bcdProc (see [bcdProc.doc.html](#)) is used mostly to clip and remove redundant data from binary cartographic data (.bcd) files. The program shp2bcd (see [shp2bcd.doc.html](#)) is mainly used to generate bcd files from shape files, but can also be used to query shape file attributes. The program test\_grhi\_remap (see [test\\_grhi\\_remap.doc.html](#)) is used to create scale specific topography images from raw topography data.

### **7.5) Programs that manipulate tables for gridded data and the Volume Browser**

The program makeGridKeyTables (see [makeGridKeyTables.doc.html](#)) is the program that generates the Volume Browser data and depictable keys based on the field, source, and plane tables. The program processStyleInfo (see [processStyleInfo.doc.html](#)) is responsible for generating the style information (such as contour intervals) for each displayable item in the Volume Browser based on the contents of the \*.rules files. The program testGridKeyServer (see [testGridKeyServer.doc.html](#)) has as its basic function serving as a unit test for the code that reads and parses the field, source, and plane tables, but it can also be used to query information from these tables. As such it is used in the creation of cdlTemplate files and directories for gridded data, and for generating lat/lon cross section depicitors and their Volume Browser menu entries. The program initCdlTemplate (see [initCdlTemplate.doc.html](#)) is used to fill empty gridded data files (created with ncgen) with geographic information and static grids, namely topography, grid spacing, and Coriolis parameter. For more information on this one is directed to [gridTables.html](#) and [styleRules.html](#).

### **7.6) Programs that create and manipulate geographic entity lookup tables**

The program [newGELTmaker](#) is responsible for generating geographic entity lookup tables (GELTs), which is how WarnGen can associate a given geographic location with a county, city, or forecast zone. The program [image\\_mask](#) is used to modify portions of one GELT based on the contents of another.

### **7.7) Programs that create station and location lists**

The program [va\\_driver](#) is the primary manner in which a meaningful progressive disclosure strategy is assigned to a list of stations or locations.

The creation of a GELT naturally creates a location list, and as such GELTs are often used as a source of information for such lists. Program [GELTtest](#), while primarily a unit test for the geographic entity lookup table software, is used in the localization to restrict the contents of station lists based on the contents of a geographic entity lookup table. Program [masterToGoodness](#) can convert a station list from a GELT into a form that can be interpreted by `va_driver`.

## 8. Realization definition files

As was mentioned before, a realization is a way of associating site specific control files with groups of localizations. By this point, enough has been said about file override and related topics that one can understand the basic functioning of a realization. Most localizations are associated with the default WFO realization. One important point about the use of realizations is that `$FXA_LOCAL_SITE` and `$FXA_INGEST_SITE` should normally both refer to localizations that use the same realization. Having the ingest and display localizations be from different realizations may work in some cases, but in general it should be avoided.

Currently, there are four valid realizations defined: 'RFC', 'NC', 'radonly', and 'cwb'. The 'RFC' realization is used to create localizations suitable for use in river forecast centers. The 'NC' realization is used to create localizations suitable for use by National Centers. The 'cwb' realization is used to create localizations suitable for use by the Central Weather Bureau in Taiwan. The 'radonly' realization is used to create localizations suitable for displaying Nexrad radar data only, occasionally being used in Alaska and Hawaii.

## 9. Customization

Until this point, all of the files and data sets that have been discussed are part of or exist in directories controlled by the default WFO-Advanced software load. What this means is that any changes made on site to the files discussed so far will go away the next time new software is delivered. Obviously, it would be desirable if users could make changes on site that would be preserved when new software was delivered. The mechanism by which this is accomplished is referred to here as customization.

There are two additional environment variables that are important to customization: `FXA_CUSTOM_FILES` and `FXA_CUSTOM_VERSION`. `FXA_CUSTOM_FILES` can point to any arbitrary directory in the file system; it could be either a local or remote mounted disk. `FXA_CUSTOM_VERSION` needs to point to a subdirectory in the directory pointed to by `FXA_CUSTOM_FILES`. By default, `FXA_CUSTOM_FILES` points to the directory `$FXA_DATA/customFiles`. Since this is a cross-mounted disk, a change made here will affect future localizations run on any machine. It is important to note that the current configuration

does not allow any customization files to directly affect the operation of the workstation or ingest software; customization files can only change the way a localization is built.

In order to make use of the customization features, the environment variable `FXA_CUSTOM_FILES` (and, optionally, `FXA_CUSTOM_VERSION`) must be set prior to running any localization that one wishes to be customized. Unlike `FXA_LOCAL_SITE` and `FXA_INGEST_SITE`, there is no way to set `FXA_CUSTOM_FILES` or `FXA_CUSTOM_VERSION` on the command line. If either are left unset, they will normally default to their previous values the last time a given site's localization was run. If one wants to rerun a site's existing localization using different values of `FXA_CUSTOM_FILES` or `FXA_CUSTOM_VERSION`, one must use the leading 'n' flag (not '-n') in the command line of `mainScript.csh`.

Customization works by extending the existing file override capabilities to also include files in `$FXA_CUSTOM_FILES` or `$FXA_CUSTOM_FILES/$FXA_CUSTOM_VERSION`. If both `FXA_CUSTOM_FILES` and `FXA_CUSTOM_VERSION` are defined, then the following file paths can also participate in file override:

```
$FXA_CUSTOM_FILES/*  
$FXA_CUSTOM_FILES/$FXA_LOCAL_SITE-*  
$FXA_CUSTOM_FILES/$FXA_CUSTOM_VERSION/*  
$FXA_CUSTOM_FILES/$FXA_CUSTOM_VERSION/$FXA_LOCAL_SITE-*
```

If append override is being used, as is very often the case with customization files, then this is the order in which they get appended. If using copy override, then this is the order in which they are copied so the last in the list that exists is what is used.

Here is an example of how the custom version might be used. Suppose one wanted to move the regional area 200 km to the south in the summer. In directory `$FXA_CUSTOM_FILES/summer`, one could create a file `mainConfig.txt`, containing the entry '@@@REGNORTH -200' (see [directives.html](#) for more information). Then all localizations would be rerun twice a year, spring and fall, with the value of `FXA_CUSTOM_VERSION` specifically set to 'summer' or an empty string, and using the 'n' flag (not '-n') to enable changing the customization environment. In the fall, `FXA_CUSTOM_VERSION` would be defined to an empty string, and in the spring it would have a value of 'summer'. What this would do is move the regional scale 200 km south of the default for the warm season and put it back to the default for the cool season. Between these two localization runs that were meant specifically to adapt to the season, other localization runs would just pick up the last specifically set value of `FXA_CUSTOM_VERSION`.

When the concept of a separate place for local site modifications was originally conceived, it was thought to be something that would be used in a very limited way. Thus, not many files were set up to be overridable from `customFiles/`. In hindsight, this was a huge blunder; allowing files to be overridable from `customFiles/` should have been the rule rather than the exception. At the current time, things have evolved to the point where most files are overridable from `customFiles/` in some fashion or another.

What follows is a list of files in \$FXA\_HOME/data or nationalData/ that are not overridable from customFiles/. An asterisk means that, while the file is not directly overridable in customFiles/ by a file of the same name, the basic functionality is overridable in customFiles/ by other means.

- \* gridSourceTable.txt
- \* gridSourceTable.template
- \* tdlSourceTable.template
- \* tdlActiveGridSources.txt
- \* tdllInactiveGridSources.txt
- scaleInfo.txt
- usTopo.dat.gz
- akTopo.dat.gz
- caribTopo.dat.gz
- pacTopo.dat.gz
- worldTopo.dat.gz
- \* dataInfo.txt
- \* depictInfo.txt
- \* dataInfo.manual
- \* depictInfo.manual
- GOESImagerInfo.txt
- SatImagerInfo.txt
- \* imageStyle.txt
- \* tdllImageStyle.txt
- ijklSatDatamenu.txt
- mnopqSatDatamenu.txt
- \* colorMaps.mark
- cccLatLon.txt
- cccLatLonPatch.txt
- wmoExceptions.txt
- natMosaicDataMenus.txt
- natMosaicProductButtons.txt
- natMosaicDepictInfo.txt
- natMosaicDataKeys.txt
- allHomeDataKeys.template
- allHomeDepictKeys.template
- tdlRadarDataMenus.template
- countyPlus.bcd
- \* wwaDefaults.txt
- \* CitiesInfo.txt
- MarineInfo.txt
- synopticStationTable.txt
- metarStationTable.txt
- maritimeStationTable.txt
- profilerStationTable.txt

raobStationTable.txt  
modelBufrStationTable.txt  
goesBufrStationTable.txt  
poesBufrStationTable.txt

# Adaptive Plan View Plotting

## Table of Contents

- [1\) Introduction](#)
- [2\) Key Entries for Plan View Plot Displays](#)
  - [2.1\) Plan View Plot Displays](#)
  - [2.2\) Sounding Displays](#)
- [3\) Design Files](#)
  - [3.1\) Overall structure of design files](#)
  - [3.2\) A design file example](#)
  - [3.3\) Local modification and testing](#)
- [4\) Lookup Table Files](#)
  - [4.1\) String to string lookup table](#)
  - [4.2\) String to number lookup table](#)
  - [4.3\) Number to string lookup table](#)
  - [4.4\) Number to number lookup table](#)
  - [4.5\) Number to byte lookup table](#)
- [Appendix 1\) Keywords](#)
  - [A1.1\) Global keywords](#)
  - [A1.2\) Item keywords](#)
  - [A1.3\) Raw data keywords](#)
  - [A1.4\) Function keywords](#)
  - [A1.5\) Display keywords](#)
- [Appendix 2\) Functions](#)
  - [A2.1\) Special functions](#)
  - [A2.2\) Conversion functions](#)
  - [A2.3\) Mathematical functions](#)
  - [A2.4\) Meteorological functions](#)
  - [A2.5\) Logical functions](#)
  - [A2.6\) List handling functions](#)
- [Appendix 3\) Display methods](#)
- [Appendix 4\) Constants with predefined meanings](#)
- [Appendix 5\) Enhancements to the netCDF files](#)
- [Appendix 6\) Standard lookup tables](#)

## 1) Introduction

Adaptive Plan View Plotting refers to an AWIPS capability that allows one to add new kinds of plan view plots or change the way existing ones look simply by changing plain text static metadata. We refer to plain text data files as *design files*.

These design files are used in several ways. They are used to make plan view plots, to help pass data from point data files to the code that creates sounding and profiler displays, and to help pass data from point data to the volume browser. This document will focus on plan view plots, sounding and profiler displays.

In order to make this work, the data to be plotted need to be in a self describing format. To this end, the plotting of all point data sets is done from netCDF files. In 5.0, all pre-existing netCDF files were enhanced by adding some additional variables and attributes. However, none of the existing variables or attributes were changed, so existing applications that read netCDF point data still work.

While there is a performance penalty for this additional flexibility, it is not prohibitive. The additional flexibility should be extremely useful for LDAD data, allowing each site to plot whatever data ends up in their LDAD netCDF files.

## 2) Key Entries for Plan View Plot Displays

Under the new paradigm, menu entries and product button entries work exactly as before to display a given depict key. Depict key and data key entries are somewhat different.

### 2.1) Plan View Plot Displays

Here is a sample depict key entry; we are using the standard METAR plot from depictInfo.manual:

```
120 |72 |82,27001,1003| |0,1003|1 |METAR |METAR Plot |1|0|1| | |900
```

Depict key entries for plan view plots include two changes. First, all entries have a depictable type (second vertical bar delimited column) of 72. This is because all of these displays use the same software module to create the display; what makes them different is the associated static metadata. Second, column three, which is the list of associated data keys, contains a new entry (27001 in this case), which points to what is referred to as a 'design file.' For the new adaptive plan view plotting, the first data key must point to the data directory, the second must point to a design file, and the third optional key must point to a static progressive disclosure file.

Here are the associated data keys from dataInfo.manual:



We will begin by discussing the overall structure of design files. Then an example of a design file will be presented and discussed in detail. Finally, we present some notes on local modification and testing.

### 3.1) Overall structure of design files

Design files are plain text files. As the text for a design file is interpreted, any line that is all spaces, blank, or begins with `//` is ignored. A backslash at the very end of a line is considered a line continuation. Each interpreted line consists of a keyword followed by one or more space-delimited arguments. (Within the remainder of this document, except in actual examples, keywords will be ``quoted'` and their arguments will be *italicized*.) A keyword and its arguments must appear on the same line; the continuation can be used to make long entries more readable. Because arguments are primarily space delimited, the user is allowed to use a tilde (`~`) to stand for a space in string arguments. In such strings, an escaped tilde (`\~`) is used to represent a tilde. When such escaping is in force, `\t` will be converted to a tab, `\n` will be converted to a carriage return, and anything else escaped will be converted to that same single character. This means that `\\` will result in a single backslash, and an argument with a single backslash will be interpreted as an empty string. Spaces are not allowed in file names, item IDs, keywords, and other reserved words, so escaping is generally ignored for these.

A design file is broken into two major sections, the *global definitions* and the *item definitions*. The global definitions are all entries that refer to the design file as a whole; the item definitions are all entries that refer to a specific item. All global definitions must appear before any item definitions, but the individual global definitions can be in any order. A complete list of keywords for both global and item definitions can be found in [Appendix 1](#).

The presence of an ``item_id'` keyword is what triggers the existence of an item. The entries that define an item span from that ``item_id'` to the entry immediately before the next ``item_id'`. ``item_id'` is always the first entry of an item definition, but within an item definition other individual entries may appear in any order. Also, the items as a whole may appear in any order.

An item is a logical entity that may contain raw data from a netCDF file, the result of a function, a constant, or a display method. Each item that contains a display method will normally also contain raw data or a function result, although there is a new feature that allows one to display constants directly in some instances. Containing raw data is mutually exclusive with containing either a function result or a constant. Functions results were previously mutually exclusive with constants, but new features allow constants to be the result of functions in some cases. Certain functions will now produce constant output if all the inputs are constant, and there are new functions that are designed to produce constant output. Another thing to note about constants is that there are certain predefined constants that are used to communicate information and control back and forth between design files and the rest of the system (see [Appendix 4](#)).

Each item is defined by its characteristics and the data it contains. Here we will focus on what these characteristics mean; how they are determined will be discussed later. One primary

characteristic of an item is its dimensionality (keyword ``dimension'`), which can be *constant*, *scalar*, or *list*. A constant item contains just one constant value, a scalar item contains one value per station or record, and a list item contains two or more values per station or record. Another primary characteristic of an item is its indexing; most items contain data on a per station basis, but some contain some data on a per netCDF file record basis. Station versus record indexing is meaningless for a constant item. A third primary characteristic is the base data type. The base data types are *float*, *int*, *short*, and *string*. It is also possible to talk of the major data type of an item, which is either *string* or *numeric*. Constants have only a major type, string versus numeric.

The most important characteristic of an individual data value is whether or not it is null. Each base data type has a corresponding well-defined null value, 9e99 for doubles, 1e37 for floats, 2147483647 for ints, 32767 for shorts, and a zero length string for strings. The null double value is important because numeric constants are held as doubles and all numeric values from lookup tables are presented as doubles. Where it is important to assign an arbitrary non-null value to a data item, 0 is used for numeric values and "0" is used for strings.

A constant item not produced from a function contains only one additional entry beside its `item_id`, with a keyword of ``constant.'` If the first argument is a single equals sign followed by another argument that is a numeric value, then the constant will be numeric. Otherwise, the first argument will be used to define a string constant. Spaces are allowed in string constants, so here a tilde should be used for a space. It is possible to override the value of a constant in a design file directly from the associated depict key entry. This is done in the 13th vertical bar delimited field, the so-called `extraInfo` field, which contains a list of comma-delimited strings. In this case, each pair of strings is assumed to be the ID of a constant item followed by an override value.

An item is identified as containing raw data primarily by the presence of a ``netcdf_id'` keyword. This keyword is followed by the name of the netCDF variable from which it reads data. The most common other keywords for a raw data item are ``type,'`dimension,'` and ``record.'` The possible arguments for the keyword ``type'` are *string*, *float*, *int*, and *short*. The reason the user supplies this information rather than it being obtained from the netCDF file is so that the validity of function relationships can be determined when the design file is parsed. The possible arguments for the keyword ``dimension'` are *scalar* and *list*. If *scalar*, there is one value for each record, meaning that variable should be dimensioned in the netCDF file as (recnum), or (recnum,strlen) in the case of a string. If *list*, then there are multiple values for each record, meaning that variables should be dimensioned (recnum,listlen), or (recnum,listlen,strlen) in the case of a string. Here recnum must refer to the UNLIMITED dimension, and listlen and strlen to any valid dimension appropriate for the variable. A recent enhancement allows a variable with no dimensions to be read in as a *scalar* of type *float*, *int*, or *short*. That single data value gets copied to each station or record to fill out the data for the item.

If the argument of the ``record'` keyword is *true*, then that item will contain values corresponding to records in the netCDF file. Otherwise, it will contain data for each station. This distinction is important because a single station may have data available at multiple times. In

order to be displayable, all data eventually needs to be station indexed--the primary reason for defining record indexed items is to exert specific control over how the single station data value is selected from several records with data from the same station. Normally the data selected will be from the observation closest to the time stamp of graphic being created. The conversion of record indexed data to station indexed data is affected by the keywords ``by_record,'` ``remove_by,'` ``indexing_table,'` ``accum_period,'` ``valid_period,'` ``average_period,'` and by the ``rec_to_sta'` function.

Because the basic station and time identification of a record in the netCDF file can involve multiple client variables (see description of `idVariables` and `timeVariables` global attributes in [Enhancements to the netCDF files](#)), special arguments to the ``netcdf_id'` keyword are available to allow access to these, namely `_ids_` and `_times_`. These variables need to be treated as *string scalar* and *int scalar*, respectively.

An item is identified as containing a function output primarily by the presence of a ``function'` keyword. This keyword is followed by the function name (see [Appendix 2](#) for a complete list of available functions). A function item must also contain an entry with an ``inputs'` keyword. The arguments that follow ``inputs'` are a list of the *item\_ids* that are input to the function. These may be other function outputs, raw data, or constants, depending on the requirements of the function. (Note that numeric strings cannot be used as function inputs; any additive or multiplicative value must be pre-defined as a constant. This is not true of [display keywords](#), for which numeric inputs are allowed, as demonstrated in the example in the next section.) There is no need to specify the data type or dimensionality of a function; that is all determined by the characteristics of the inputs and the function used.

An item is identified as having a display method primarily by the presence of the keyword ``placement.'` The kind of display is determined by the argument to the keyword ``method.'` For some methods, the placement is predetermined by the method and the argument to the keyword `placement` is just *free*. Otherwise it is one of nine predefined placement positions (*upper\_left*, *center*, etc.). While most display methods require only one input, some require multiple inputs, in which case a special function called ``gather'` is used to hand multiple items to a single display method. The reader is directed to [Appendix 3](#) for a complete list of display methods. The reader should be aware that while some display keywords such as ``multiplier'` look as though they could be used to change the data values contained in an item, they cannot. All display keywords can do is change how data is displayed. A function must be used to change the data values contained in an item.

### 3.2) A design file example

What follows is a sample design file. Again we use the standard METAR plot (`nationalData/metarStdDesign.txt`) as an example.

#### Annotated metarStdDesign.txt

```
// Standard metar plot
```

```
size 45
time_step 3600
//stations_path metar_stations.
```

In this example the last global definition is "time\_step 3600" and the first item definition is "item\_id RT." The entry "size 45" specifies that each plot model is to be treated as 45 pixels in size for the purpose of calculating progressive disclosure. The entry "time\_step 3600" means that each graphic of this type in a time sequence will be separated by one hour, which is 3600 seconds. All time periods in design files are in units of seconds.

```
item_id RT
type string
dimension scalar
record true
indexing_table rank_report_type.txt
netcdf_id reportType
```

Normally, when the code figures out which data record to copy to a station, the record with a time closest to the valid time of the whole data set is used. Here, the item *RT* is used to specify that we prefer to use observations with a report type of METAR if available, instead of a SPECI that may be closer to the data set time. To make this work, this item is specified to be a record item and an entry with the keyword ``indexing_table'` is supplied. The file *rank\_report\_type.txt* contains the data for what is called a [lookup table](#). This particular table is a string to number lookup table, and is used to assign arbitrary rankings to the strings METAR and SPECI.

```
item_id TXT
type string
dimension scalar
netcdf_id rawMETAR
sample true
```

This item is reading in the raw text of the METAR. Since the keyword ``sample'` is there with the argument of *true*, this text is what is presented for sampling. In order to sample, an item must be a scalar string type with station indexing.

```
item_id CLOUDS
type string
dimension list
netcdf_id skyCover
```

This item pulls in the cloud information but does not directly display it. That will be done by a later item (and note that the display does not have to be done by the immediately following item). This item has a *list* argument to its ``dimension'` keyword because there can be multiple cloud decks.

```
item_id DPTYP
```

type short  
dimension scalar  
netcdf\_id pressChangeChar  
item\_id DPVAL  
type float  
dimension scalar  
netcdf\_id pressChange3Hour  
item\_id DP3  
function gather  
inputs DPVAL DPTYP  
method trend  
placement right  
multiplier 0.1  
format %3.2d

DPTYP and DPVAL are used to pull in the characteristic and amount of the pressure change. DP3 uses the *gather* function, which is a special function used only to pass one or more previously defined items to a display method. The display method here is *trend*, which is specifically tailored for the task of displaying pressure change data. In the design of the capability, we have tried to avoid specifically tailored functions or display methods, but a few exist. Note that the units conversion from pascals to tenths of millibars is done with the generic 'multiplier' keyword rather than being folded into the method.

item\_id SPD  
type float  
dimension scalar  
netcdf\_id windSpeed  
item\_id DIR  
type float  
dimension scalar  
netcdf\_id windDir  
item\_id GUST  
type float  
dimension scalar  
netcdf\_id windGust  
item\_id WIND  
function gather  
inputs SPD DIR GUST  
method barb  
placement free  
multiplier 1.944

SPD, DIR and GUST are used to pull in the wind speed, wind direction, and gust speed, respectively. The item WIND uses *gather* to present these to the *barb* method. Note that since this method predetermines the placement of what is being plotted, *free* placement is used. The multiplier is used to convert from meters per second to knots.

item\_id WX  
type string

```
dimension scalar
netcdf_id presWeather
placement left
method translation
table_file wx_symbol_trans.txt
alt_char_set weather
```

This is an example of a raw data item that is immediately handed to a display method. The method *translation* specifies that a [string to string lookup table](#) is expected, and the recursive translation method of that table will be called to produce the text for output. The ``table_file'` keyword is used to supply the name of the file containing the lookup table. All text that cannot be translated will be shown as plain ASCII, but the text that is translated will be displayed using the [weather character set](#).

```
item_id TC
type float
dimension scalar
netcdf_id temperature
item_id TdC
type float
dimension scalar
netcdf_id dewpoint
item_id T10
type float
dimension scalar
netcdf_id tempFromTenths
item_id Td10
type float
dimension scalar
netcdf_id dpFromTenths
item_id T
function or
inputs T10 TC
placement upper_left
method formatted
multiplier 1.8
offset -459.67
format %d
min_trans -60
max_trans 130
```

TC is used to pull in the value decoded from the regular temperature field in the METAR, which is in degrees Celsius. U.S. METARs also sometimes have a temperature in tenths (used to recover the precision of the Fahrenheit observation) in the remarks section; the T10 item is used to pull that in. Then the T item pulls these two in using the *or* function and displays them. The order is important: by putting T10 first, it will use the T10 value if available, and revert to using the TC value otherwise. These kelvin temperatures are converted to Fahrenheit using the multiplier and offset of 1.8 and -459.67. The ``min_trans'` and ``max_trans'` keywords are used to supply sanity check limits to the results of the units conversion. A completely analogous thing

happens with the items TdC, Td10, and Td (below) in order to display the dewpoint.

```
item_id Td
function or
inputs Td10 TdC
placement lower_left
method formatted
multiplier 1.8
offset -459.67
format %d
min_trans -60
max_trans 100
item_id SLP
type float
dimension scalar
placement upper_right
method formatted
multiplier 0.1
netcdf_id seaLevelPress
format %5d
trim_count 2
min_trans 8000
max_trans 11000
```

This is yet another example of handing raw data directly to a display method. The multiplier of 0.1 turns pascals to tenths of millibars, and `%5d` is used to format that result. Keyword ``trim_count'` is followed by the number of characters to strip off the front of the formatted string before displaying it. This is how we follow the convention of not displaying the hundreds of millibars in the pressure field.

```
item_id VV
type float
dimension scalar
netcdf_id vertVisibility
item_id OBSstr
constant OBS
item_id OBS
function and
inputs OBSstr VV
```

What happens here is that `OBSstr` is defined as the string "OBS". Then, if `VV` is non-null, the "and" of `OBSstr` and `VV` will be that same string (because the first item is passed forward). Thus, the item `OBS` will be either null or "OBS". This is used below as input to the `CEILING` definition.

```
item_id CLGDECK
function select
function_table cloud_select.txt
inputs CLOUDS
```

This is a list item; *select* is used to choose one value from the list, based on the CLOUDS field assigned above. The file named as the argument to the ``function_table'` keyword is a string to number table that is used to assign arbitrary rankings to each cover type; the cover type with the lowest ranking number is used.

```
item_id CEILING
function or
inputs OBS CLGDECK
placement center
method lookup
table_file cloud_chars.txt
alt_char_set special
```

CEILING will be OBS (string "OBS" - obscured) if that's been set, else CLGDECK determined above. The display method used is *lookup*, which means use a lookup table to do a simple translation (not the recursive one done for WX) to produce the output text. The table used is a string to string table defined in `cloud_chars.txt`, and the result is displayed using the [\*special character set\*](#).

The standard METAR design file retrieves many of the netCDF variables stored by the decoder. Most of the other variables are used by one or another METAR plots, which you can see by examining other design files in `/awips/fxa/data/localization/nationalData`. A companion document, [METARElements.html](#), provides additional examples of how to access and display the remaining variables.

### 3.3) Local modification and testing

As noted, default design files delivered with an installation reside in `nationalData/`. Suppose you want to change the profiler perspective plot to use arrows instead of barbs. By examining `dataMenus.txt` (button 3998), `depictInfo.manual` (3998 uses design file 27109), and `dataInfo.manual`, you'll find that the design file for this product is `profPerspPlotDesign.txt`. You can put a copy of this file in your `localizationDataSets/<LLL>` directory, then modify it and see the results. A caveat is that the design file is read only when the display first needs it. Thus, you have five chances to see the effect of changes (by swapping in side panes). Once you've used up all the IGC processes, you'll need to restart D2D, or you can kill one or more IGC\_Processes to get a fresh canvas. (New IGC(s) should start automatically, but you can use Options->Restart Dead Panes... if necessary.)

## 4) Lookup Table Files

As mentioned before, lookup table files are a means by which a user can perform arbitrary data conversions. Lookup table files are plain ASCII files, with blank lines and those that begin with ``/'` ignored, just as with design files. There are five types of lookup table files. The first line of any lookup table file is a three character string that designates its type. All of the default lookup

table files exist in nationalData/ and have file names that look like \*\_\*.txt. Just like design files, they are subject to copy override from realization, site-specific, and customization files. Since space-delimited parsing is used in these files, one needs to use a tilde (~) to designate the existence of a space in an input string. An escaped tilde (\~) can be used to designate a tilde.

A new feature is that one can create a lookup table object based on text directly in a design file. See the description of the ``inline_table'` keyword in [Appendix 1.1, global keywords](#).

See [Appendix 6](#) for a listing of standard lookup table files.

#### **4.1) String to string lookup table**

The first line of a string to string lookup table is "s2s". A string to string lookup table is used to convert some arbitrary string to some other string. Most entries are a line with a lookup string followed by a result string to which to translate it. Sometimes, there is a need to have characters in the result string that are unprintable in ASCII. In that case, the result string can be expressed with a colon followed by a list of integers designating the character codes (all space delimited). It is also possible to make an entry with only a lookup string, which means that the result of translating that string will be an empty string.

There are four special keywords that are recognized in a string to string table. Once a line has been encountered that does not have one of these keywords on it, further occurrences of these keywords will be treated as a regular lookup string. Normally, if an attempt is made to translate an input string that does not exist in the table as a lookup string, the output will be some constant default string. The keyword ``pass,'` if present, indicates that if the input string is not available as a lookup string, then the resulting translation should be the input string. The usual default string is an empty string, but using the keyword ``default'` allows one to supply a different default string. There is a special type of translation available for the string to string table that will attempt to translate all possible substrings in the input string, as well as the entire string. The keywords ``left'` and ``right'` allow one to perform an edit operation on the input string before it is translated in this case. The arguments after the ``left'` or ``right'` keyword are a lookup and result string expressed exactly as in a regular entry. For ``left,'` the first occurrence of the corresponding lookup string is located, and all text up to that occurrence is replaced with the corresponding result text. ``right'` works analogously with the last occurrence and the end of the string to translate.

#### **4.2) String to number lookup table**

The first line of a string to number lookup table is "s2n". Such a lookup table is used to convert some arbitrary string to some numeric value. Most entries are a line with a lookup string followed by a numeric value to which to translate it. Internally, numeric values in a string to number table are held as doubles.

There are three special keywords that are recognized in a string to number table. Once a line has been encountered that does not have one of these keywords on it, further occurrences of these keywords will be treated as a regular lookup string. If an attempt is made to translate an input string that does not exist in the table as a lookup string, the output will be some constant default value. Normally, this default value is 9e99, universally recognized by the PlotDesign class as a null double value. Using the keyword `default` allows one to supply a different default numeric value. There is a special type of translation available for the string to number table that will attempt to scan all possible substrings for a possible lookup string, as well as the entire string. The keywords `left` and `right` allow one to perform an edit operation on the input string before it is translated in this case. The arguments after the `left` or `right` keyword are a lookup string and a translation string. For `left`, the first occurrence of the corresponding lookup string is located, and all text up to that occurrence is replaced with the corresponding translation string. `right` works analogously with the last occurrence and the end of the string to translate.

#### **4.3) Number to string lookup table**

The first line of a number to string lookup table is "n2s". An n2s lookup table is used to convert some arbitrary numeric value to a string. Most commonly, entries are a single number or a pair of numbers (representing a range), space delimited, followed by a result string. If there are overlapping ranges, order is important because the first range that matches will be used. Sometimes, there is a need to have characters in the result string that are unprintable in ASCII. In that case, the result string can be expressed with a colon followed by a list of integers designating the character codes (all space delimited). It is also possible to make an entry with only one or a pair of numbers, which means that the result of that lookup will be an empty string. Internally, numeric values in a number to string table are held as doubles.

There is one special keyword that is recognized in a number to string table. Normally, if an attempt is made to look up a number that does not exist in the table or is outside all specified ranges, the output will be an empty string. Using the keyword `default` allows one to supply a different string to be output in the case of a failed lookup.

#### **4.4) Number to number lookup table**

The first line of a number to number lookup table is "n2n". This type of table is used to convert some arbitrary numeric value to another numeric value. Most commonly, entries are a list of two, three, or four numbers, space delimited. Two numbers means translate the first number to the second. Three numbers means that if the input number falls between the first two numbers, the result is the third number. Four numbers means that if the input number falls between the first two numbers, do an interpolation to the range represented by the last two numbers. If the first number is less than the second, this interpolation will be linear, otherwise the interpolation will be logarithmic. For the logarithmic case, the first and second number must be non-zero and the same sign.

There are two special keywords that are recognized in a number to number table. Normally, if an attempt is made to look up a number that does not exist in the table or is outside all specified ranges, the output will be 9e99, universally recognized by the PlotDesign class as a null double value. Using the keyword `default` allows one to supply a different number to be output in the case of a failed lookup. If keyword `pass` is present, then a failed lookup results in the number input to the lookup being passed through.

#### **4.5) Number to byte lookup table**

**Note:** This table type currently is not used within the framework of the adaptive plan view plotting -- it is used by the GribImgDecoder process to convert floating point values from GRIB files to byte values for storage as images and for processing the radar DPR product. It's documented here only because this is where all the other lookup table types are documented.

The first line of a number to byte lookup table is "n2b". This type of table is used to convert some arbitrary numeric value to a byte value.

Entries are one or two floating point numbers followed by one or two integer (byte) values from 0 to 255, all space delimited. When only one byte value is present, either the single floating point value or the range of floating point values gets converted to the byte value specified. If there are two byte values, there must be two floating point values, in which case an interpolation is done from the range of first pair to the range of the second. If the first number is less than the second, this interpolation will be linear, otherwise the interpolation will be logarithmic. For the logarithmic case, the first and second number must be non-zero and the same sign.

There are two special keywords that are recognized in a number to byte table. Normally, if an attempt is made to look up a number that does not exist in the table or is outside all specified ranges, the output will be 0, universally recognized by the image display software in AWIPS as a null image value. Using the keyword `default` allows one to supply a different byte value to be output in the case of a failed lookup. For efficiency, internally within this class, a single lookup array is created covering the entire range of floating point values supplied. The floating point resolution of this table is the value following the special keyword `resolution`. This will default to 1.0 if this keyword is not supplied. It is very important that the internal resolution of the table be well chosen. Too coarse, and lookups will not be accurate; too fine, and the table will take up more memory than necessary.

---

### **Appendix 1) Keywords**

By convention, keywords are all lower case with underscores. This discussion will divide keywords into five categories: global, item, raw data, function, and display. Unless otherwise

mentioned, the reader should assume that each keyword takes one and only one argument. The reader should also note that all time periods in design files are in seconds.

All global keywords must appear in the design file before any keywords from other categories. Global keywords refer to the file as a whole; all other keyword are specific to their corresponding item.

### A1.1) Global keywords

Keyword	Argument
size	<p>The assumed size in pixels of each station model plot. This is used for progressive disclosure calculations.</p>
shared_data	<p>If the argument to this keyword is <i>true</i>, then all depictables that use this design file, access data from the same directory, and have the same frame time, will share the raw data they read off the disk. This can help the speed of loading when several overlays are all loading the same large, common, data set.</p>
time_step	<p>The number of seconds separating the valid times of each displayable frame. For example, 3600 would be one hour; thus a graphic would be created with valid times on the hours. By default, this could potentially contain observations ranging in time from 30 minutes before to 30 minutes after the hour.</p> <p>time_step should be greater than or equal to filePeriod divided by nInventoryBins, as defined in the <a href="#">CDL</a>. Alternatively, a value of one means displays will be created for individual record times (should be used with caution).</p> <p>Zero is a special case that means that the display inventory will be assumed to be exactly the same as the file time stamps. This optimization is currently used only for the DMD display.</p> <p>A negative value is a flag always to use the simplest inventory algorithm possible. This should be done only if it is OK for the inventory not to take into account the availability of data sets not in the primary time bin that nonetheless are included in creating the final display.</p> <p>An optional second argument is the number of seconds to offset each frame time from an even multiple of the time step; this will normally be from minus one half to plus one half the time step. This is not the same as `data_offset,' which modifies the data gathering window, not the frame time.</p>
default_period	<p>&lt;default&gt; The display will accept data from a time bin the same size as the value of `time_step.'</p> <p>&lt;number&gt; The size of the time bin over which data will be allowed to appear in the graphic.</p> <p>The bin is always centered at the valid time assigned to the display unless a</p>

non-zero value of `data_offset` is given.

`inline_table` Typically, a lookup table refers to a separate file. If this keyword is present, then the argument is the name of an in-line lookup table, which must be unique within the design file. All text that immediately follows, up to the next legitimate keyword, is considered to be the definition of the lookup table known by that name. A table defined in this manner can be accessed only within this design file, and it can override a lookup table file of the same name.

`<default>` Data will be gathered primarily for a time period centered on the valid time for which the frame is created.

`data_offset` `<number>` If given, center of primary data gathering period is offset from frame valid time by this much.

This keyword is similar to the item keyword `time_offset`, except that it will never feed back to inventories. It is different from the second argument to `'time_step,'` which modifies the frame time, not the data gathering window.

`<default>` Normally, only one record among several with the same station ID will be selected for display.

`by_record` `true` All records in the selected default time bin will be made available for display regardless of whether they have unique station IDs. When this is in effect, the use of the item keywords `'time_offset,'` `'accum_period,'` or `'valid_period'` is no longer legal.

`predefined_only` `<default>` Any station for which no static progressive disclosure entry is available will plot if its latitude and longitude can be found in the netCDF file. In this case, these stations will never display at a zoom level lower than nearby neighbors that do have static progressive disclosure entries.

`true` Only stations appearing in the static progressive disclosure file provided will plot.

This keyword is used to control whether and how progressive disclosure is calculated on the fly based on which data is available. One can divide stations to plot into two categories, those with static progressive disclosure and those without.

`dyn_prog_disc` `<default>` Those stations with static progressive disclosure will retain it. Those without will have their progressive disclosure dynamically computed if there are not too many stations, otherwise they will all be assigned some small constant value that guarantees they will show up only at very high zooms or densities.

`none` Stations missing progressive disclosure will always be assigned the small constant value.

`missing` Stations missing progressive disclosure will always have progressive disclosure dynamically computed.

	<i>all</i>	All stations, including those for which static progressive disclosure information was available, will have their progressive disclosure dynamically computed based on which stations actually have data.
<i>by_density</i>		If the argument to <code>`by_density'</code> is false, the progressive disclosure of stations will not respond to the density control, but will still respond to magnification and zooming. In conjunction with the item keyword <code>`min_density,'</code> this can be used to create a plot where the density is primarily used to control how many data items are plotted for each station, rather than how many stations plot.
<i>stations_path</i>		A fragment of a file name to which the list of stations rendered will be written each time a frame is drawn. It is written into the <code>localizationDataSets/&lt;LLL&gt;/</code> directory and has an extension with the process ID encoded.
<i>hard_accums</i>	<i>record</i>	when performing an accumulation in order to convert data from <code>&lt;default&gt;</code> record to station indexing, as long as at least one non-undefined value is available, a value will be created for the accumulation there must have been a record available for each valid period that made up the accumulation
	<i>true</i>	there must have been a non-undefined value available for each valid period that made up the accumulation
<i>sample_format</i>		Normally, when sample text is output, whatever text is in the item designated as being the sample item is just written to the screen as is. <code>`sample_format'</code> can take one or more arguments, each being a separate line of formatting information. The standard <code>~</code> is used as a place holder for a space. As the text to be formatted is scanned, each <code>%c</code> results in the next character being used in the output, and each <code>%s</code> results in the next space-delimited word being used in the output. If the last format specification is <code>%s</code> , then that format specification will be replaced with whatever text is left regardless of spaces. If in the course of formatting, the text to be formatted is exhausted, the rest of the format specifications will be replaced with empty strings. Finally, one can cause a string with spaces to be treated as a single item if it is enclosed in a pair of double quotes ( <code>"</code> ). If one wants a double quote to be output, one can escape it with a backslash ( <code>\</code> ).
<i>diag_dump</i>		A numerical argument, causing diagnostics to be written to <code>stderr</code> as the design file is parsed and used to obtain data. <code>&gt;=10</code> minimal diagnostics <code>&gt;=20</code> detailed diagnostics <code>&gt;=30</code> diagnostics that include information about individual records and stations processed A ones place of 9 will result in outputting the contents of all the data items.

## A1.2) Item keywords

Keyword	Argument
item_id	<p>The presence of this keyword triggers the existence of an item. The entries that define an item span from its `item_id` to the entry immediately before the next `item_id.` This keyword is always the first entry of an item definition, but within an item definition other individual entries may appear in any order.</p>
sample	<p>This keyword must be applied to an item that has a scalar string type and is station indexed. When present with an argument of <i>true</i>, it will cause the text from this item to be used for sampling. When the value is a positive integer, this will also invoke sampling based on this variable, where the number is the maximum number of stations for which output can be presented at the same time. This feature allows the user to see sample text from more than just the one station nearest the cursor in the case where multiple stations might be very close together and hard to distinguish graphically. This keyword should be used on only one item. A second optional argument controls how sampling responds to progressive disclosure or the elimination of stations for plotting using the <code>remove_by</code> keyword; normally all stations are samplable regardless of these considerations. A value of <i>now</i> means sample only stations currently visible, <i>ever</i> means sample only stations that one could eventually see by zooming.</p>
remove_by	<p>This keyword must be applied to an item that has scalar dimensionality. All basic data types, string, float, int and short, have a well defined null value. When present with an argument of <i>true</i>, it will cause all records/stations for which this item has a null value to be removed from consideration for plotting. This keyword should be used on only one item that is station indexed, and only on one item among a group of record indexed items with the same <code>time_offset</code></p>
indexing_table	<p>This keyword must be applied to an item that has scalar dimensionality and is record indexed. This allows the value of the data item to be taken into account when selecting which of multiple records applying to the same station should be selected for display. The argument must be the filename of a lookup table with numeric output and input appropriate to the item. This table is used to convert values in this item into arbitrary rankings; lower ranking numbers are most preferred for display. This keyword should be used on only one item.</p>
goodness_table	<p>This keyword must be applied to an item that has scalar dimensionality and is station indexed. This allows the value of this data item to be taken into account when calculating progressive disclosure. The argument must be the filename of a lookup table with numeric output and input appropriate to the item. This table is used to convert values in this item into goodness values; higher goodness values are most preferred for display. The reader is directed to <a href="http://va_driver.doc.html">va_driver.doc.html</a> for more information on goodness values.</p>
accum_period	<p>This keyword must be applied to an item that is station indexed. Furthermore, it should be applied only to an item that is also raw data or is the result of a call</p>

to the `rec\_to\_sta` function. The value associated with the `valid\_period` keyword for this item directly affects the result of using this keyword. When a non-zero value for the accum\_period keyword is present (0 is the default), then an alternate method is used to map record indexed data to station indexed data for the associated item. An attempt is made to add up values over a period in time equal to the accum\_period, assuming each individual value represents the state over a period in time equal to the valid\_period. The number of record indexed values used in the summation is essentially the accum\_period divided by the valid\_period. If that number is greater than one, then this results in values being added up over some period of time. If there is only one valid\_period within the accum\_period, then this routine can be used to find a non-undefined value amongst several record indexed values irrespective of what the default record to station mapping may be. The user should note that while normally data is gathered from a time window centered on the time\_offset value for the item, accumulations are done over a period ending at the time\_offset value.

Use of this keyword is not allowed if the `by\_record` feature is used.

This keyword must be applied to an item that is station indexed. Furthermore, it should be applied only to an item that is also raw data or is the result of a call to the `rec\_to\_sta` function. The value associated with the `valid\_period` keyword for this item directly affects the result of using this keyword. When a non-zero value for the average\_period keyword is present (0 is the default), then an alternate method is used to map record indexed data to station indexed data for the associated item. An attempt is made to average the values over a period in time equal to the accum\_period, assuming each individual value represents the state over a period in time equal to the valid\_period. The number of record indexed values used in the average is essentially the average\_period divided by the valid\_period. The user should note that while normally data is gathered from a time window centered on the time\_offset value for the item, time averaging is done over a period ending at the time\_offset value.

average\_period

Use of this keyword is not allowed if the `by\_record` feature is used.

Defaults to the value of the `default\_period` keyword, which defaults to the value of the `time\_step.` It is meaningful to specify a value for the valid\_period keyword only when a value has also been supplied for the `accum\_period` or `average\_period` keyword for the purpose performing an accumulation or time average. For individual records closer in time than this, only one will be considered for accumulation or averaging.

valid\_period

A "constant" item contains only one additional entry beside its item\_id, with a keyword of `constant.'

constant

= <number> the constant will be numeric

<otherwise> A string constant. Spaces are allowed in string constants, using

the standard tilde as place holder.

### A1.3) Raw data keywords

Keyword	Argument
	An item is identified as containing raw data primarily by the presence of this keyword. The keyword is followed by the name of the netCDF variable from which it reads data.
netcdf_id	There are two special values for this keyword: <i>_times_</i> which represents an <i>int scalar</i> variable corresponding to the primary UNIX time of each record, and <i>_ids_</i> which represents a <i>string scalar</i> variable corresponding to the full, perhaps multivariable, station ID for each record.  <i>true</i> the item will contain values corresponding to records in the netCDF file  <otherwise> the item will contain data for each station
record	There are two main reasons to specify that a data value is record indexed. First, keywords <code>`remove_by'</code> and <code>`indexing_table'</code> perform functions that require a record indexed item. Second, a user may want to perform some calculations on a variable before doing the record to station conversion for display.
type	Used to specify the base data type of a raw data variable, possible arguments are <i>string</i> , <i>float</i> , <i>int</i> , and <i>short</i> . The reason the user supplies this information rather than it being obtained from the netCDF file is so that the validity of function relationships can be determined when the design file is parsed.  there is one value for each record, meaning that variable should be <i>scalar</i> dimensioned in the netCDF file as (recnum), or (recnum,strlen) in the case of a string
dimension	there are multiple values for each record, meaning that variables should be <i>list</i> dimensioned (recnum,listlen), or (recnum,listlen,strlen) in the case of a string.  Here recnum must refer to the UNLIMITED dimension, listlen and strlen to any valid dimension appropriate for the variable.
subset	Raw data read from a netCDF file must have the UNLIMITED dimension as its first dimension. Normally beyond that, there can be only one more dimension on the variable (two if string). <code>`subset'</code> allows one to read data with more dimensions by fixing some dimensions and letting others float. For any item, the subset keyword must have an argument for each dimension beyond the UNLIMITED dimension. An argument of -1 allows a dimension to float, and an argument zero or greater fixes the dimension at that index. Considering numeric variables, if all dimensions are fixed by a <code>`subset'</code> keyword, then the item becomes scalar, and if all but one dimension is fixed it becomes list. For string data, one floating dimension means scalar, two means list.

time_offset	Normally, records for an item come from a time period centered at the time of the graphic being created. Using this keyword, it is possible to specify an arbitrary offset for the center of that time period for any item. For example, a time_offset argument of -86400 would allow one to read data one day previous to the time of the graphic. Use of this keyword is not allowed if the `by_record` feature is used.
min_valid	For numeric types, any value read that is less than the argument will automatically be converted to the null value appropriate for the base type. This keyword has no meaning for strings.
max_valid	For numeric types, any value read that is greater than the argument will automatically be converted to the null value appropriate for the base type. This keyword has no meaning for strings.

#### A1.4) Function keywords

Keyword	Argument
function	An item is identified as containing a function output primarily by the presence of this keyword. The argument is a function name. (See <a href="#">Appendix 2</a> for a complete list of functions.)
inputs	Every item that has a `function` keyword must also have an `inputs` keyword. The argument to this keyword is a list of item IDs that represent the data input to the function. Depending on the function being used, order is usually very important.
recompute	Normally, once data are read into a depictable, all the functions are computed and that is that. If the argument to this keyword is <i>true</i> , then the result of this function will be recomputed every time one zooms or pans. It makes sense to set <i>recompute true</i> for any use of the `visible` function, as well as anything computed from that item.
function_table	Some functions may need a lookup table to do their work and others may optionally be able to apply one. For any such functions, the argument to this keyword is the name of that table.
parameters	One or more strings, the interpretation of which is dependent on which function is being used. A function is set up to take parameters when there is a desire to change the behavior of a function in a way that is not suitable to do with arguments, but does not result in a behavior that is fundamentally different enough to warrant an entirely different function. Parameters are only rarely required to make a function work; the code will nearly always provide meaningful defaults.

#### A1.5) Display keywords

Keyword	Argument(s)
---------	-------------

placement	An item is identified as having a display method primarily by the presence of this keyword. For some methods, the placement is predetermined by the method, and at other times the user may decide to supply specific coordinates. In these cases, the argument is <i>free</i> . Otherwise it is one of nine predefined placement positions: <i>upper_left</i> , <i>top</i> , <i>upper_right</i> , <i>left</i> , <i>center</i> , <i>right</i> , <i>lower_left</i> , <i>bottom</i> , and <i>lower_right</i> .
method	The argument determines the manner in which the data in the item are to be displayed. Most methods require only one input; some require multiple inputs, in which case special function <code>`gather'</code> is used (see <a href="#">Section A2.1</a> ). A complete list of display methods is found in <a href="#">Appendix 3</a> .
location	A pair of coordinates that are in units of numbers of characters offset from the center of the station model, right and up being positive. These coordinates need not be integers. The value of the <code>`placement'</code> keyword must be <i>free</i> for these coordinates to be used.
format	Some display methods require a C format specification, supplied here. When formatting string values, the core of the format must be a <code>%s</code> type. When formatting numeric values, the core of the format can be <code>%d</code> , <code>%f</code> , or <code>%g</code> type. The default format is dependent on the data type, <code>%s</code> for string, <code>%g</code> for float, and <code>%d</code> otherwise.
complex_fmt	This means of formatting text will allow text strings in the input data to be treated as space delimited arguments instead of a single data item. This means of formatting kicks in after the <code>`format'</code> keyword argument is used to produce a string, regardless of the data type. The argument to a <code>`complex_fmt'</code> item is similar to the one for <code>`sample_format'</code> except that it must be a single argument and cannot produce multiple lines of output. The standard <code>~</code> is used as a place holder for a space. As the text to be formatted is scanned, each <code>%c</code> results in the next character being used in the output, and each <code>%s</code> results in the next space-delimited word being used in the output. If the last format specification is <code>%s</code> , then that format specification will be replaced with whatever text is left, regardless of spaces. If in the course of formatting, the text to be formatted is exhausted, the rest of the format specifications will be replaced with empty strings. Finally, one can cause a string with spaces to be treated as a single item if it is enclosed in a pair of double quotes ( <code>"</code> ). If one wants a double quote to be output, one can escape it with a backslash ( <code>\</code> ).
trim_count	The number of characters to strip off the front of a formatted string before displaying it. It defaults to zero - trim no characters.
multiplier	<code>`multiplier'</code> works hand in hand with <code>`offset'</code> to allow units conversions to be performed on numeric values before displaying them. Numeric values are multiplied by the <code>`multiplier'</code> argument and then the <code>`offset'</code> argument is added before formatting for display. The default value is one.
offset	<code>`offset'</code> works hand in hand with <code>`multiplier'</code> to allow units conversions to be performed on numeric values before displaying them. Numeric values are

	multiplied by the <code>`multiplier'</code> argument and then the <code>`offset'</code> argument is added before formatting for display. The default value is zero. This keyword is not applicable for units conversion with vector display.
<code>min_trans</code>	A lower bound for a final sanity check on a value to display after it has gone through the units conversion check. This is not applicable for vector display.
<code>max_trans</code>	An upper bound for a final sanity check on a value to display after it has gone through the units conversion check. For vector display, this is applied to the magnitude.
<code>table_file</code>	the name of the file containing the data for a method-required lookup table
<code>undef_string</code>	By default, when a display method is given a null value to display or the value fails the final sanity check, nothing will be displayed. Here the user can supply a string that should be displayed in that case.
<code>magnification</code>	A magnification factor for whatever is being plotted. The default is one and non-integral values are meaningful.
<code>min_density</code>	Normally, individual items in a station model will always plot as long as the data are there and the station plots. The single float argument to <code>`min_density'</code> allows one to set a minimum density threshold for plotting an individual item. In conjunction with the global keyword <code>`by_density,'</code> this can be used to create a plot where the density is primarily used to control how many data items are plotted for each station, rather than how many stations plot.
<code>alt_char_set</code>	Allowable arguments are <i>ascii</i> , <i>large_ascii</i> , <i>weather</i> , <i>special</i> , and <i>large_special</i> , specifying the character set to use to display text; the default is <i>ascii</i> . These character sets are described and illustrated in <a href="#">characterSets.html</a> .
<code>attributes</code>	An arbitrary integer meant to control how certain items are drawn. Currently meaningful only to the <i>circle</i> and <i>polyline</i> methods.

## Appendix 2) Functions

The data type of a function item is determined by the specific function used and the inputs. With some exceptions, most functions will take any base data type as input. There are some broadly applicable rules about compatibility among inputs when there are multiple inputs to a function. One can never mix record indexed and station indexed items, nor can one mix record indexed items that have different values for the ``time_offset.'` With rare exceptions, one cannot mix list inputs that have different list sizes. However, one is usually free to mix scalar, list, and constant items as inputs, with the exception that it is usually not meaningful to have all inputs be constant. When mixing input items of different dimensionality, the result has the largest dimensionality. Inputs to functions can be raw data or other function results.

Here we will divide functions into six broad categories; special, conversion, mathematical, meteorological, logical, and list handling.

### A2.1) Special functions

Function Name	Description
gather	<p>This function exists solely for presenting a list of items to a display method. The most common use is to present multiple items to a method that needs more than one item, such as displaying vectors. Another use is to pass a single item to an additional display method when the item as defined already has a display method. <code>`gather'</code> cannot take constant or record indexed inputs, and only rarely takes list inputs. The specifics of what is allowed is dependent on the particular display method used (see <a href="#">Appendix 3</a>).</p>
rec_to_sta	<p>This is normally used to convert one record indexed item to a station indexed item. It is the only function that can generate a result with an indexing type different from its inputs. If <code>`accum_period'</code> or <code>`average_period'</code> is not present, it will use whatever default record to station mapping has been established for the <code>time_offset</code> of the item. If <code>`accum_period'</code> or <code>`average_period'</code> is present, then an accumulation or time average will be performed, depending on the argument values for those keywords and <code>`valid_period.'</code> If an optional second input item is provided, then the selection of which records in the first item are used to compute the output station indexed data is controlled by which records in the second item have non-null values. If there are two input items and a parameter value of <code>`max'</code> (or <code>'min'</code>) present, then the record in the first item copied into the station item will be the one corresponding to the record containing the maximum (minimum) data value in the second item.</p>
sta_id	<p><code>`sta_id'</code> takes no arguments. The output is a scalar station indexed item of string base type, which contains the ID of each station as obtained from static metadata. The fact that this function exists does not prevent one from reading the station ID from a netCDF variable.</p>
visible	<p><code>`visible'</code> takes no arguments. The output is a scalar station-indexed item of int base type, which will contain 1 if the station is visible in the current zoom state and null if not.</p>
to_const	<p>The purpose of this function is to convert any single non-constant item to a constant. By default, it returns a constant of the same major type containing the first non-null item in the data. One can optionally provide one or two additional numeric constant arguments. One constant means process only that record/station index to determine the ouput, two means process that range of record/station indices. One optional parameter can be provided, interpreted as follows. <i>last</i> means return the last non-null item found instead of the first. <i>catenate</i> should be used only with strings, and results in catenating all the strings in the input item into a single string. <i>total</i> and <i>number</i> mean return a numeric constant that is a count of how many non-null items are found; if a problem is encountered (all nulls input, missing input, optional index out of range) <i>total</i> will return a 0, <i>number</i> will return a null. <i>begin</i> and <i>end</i> mean return a numeric constant that is the station/record index of the first/last non-null item found.</p>

This routine normally takes two or more inputs of mixed types and dimensionality, the result being an int containing a count of the number of non-null items provided. If there is just one list input, then the result is a scalar containing how many non-nulls were in the list. If there is one scalar input, then the count of non-nulls is done in record/station space and copied to each item in the output.

`count`

`from_file` 'from\_file' takes no arguments, but does take two parameters. The result is a constant. The first input is either *string* or *numeric*, determining the type of the constant, and the second parameter is the name of the file from which to read the value of the constant. The file is found through the InfoFileServer. An optional third parameter is a tag with which the line in the file from which to read the constant must begin. An optional fourth parameter is the default value to use if the file and/or tag cannot be found. If one wants a default but no tag supply a single space (using a tilde) for the tag parameter.

`state_check` This is a very specialized function, with three or more inputs. The first one or more inputs are items to check against a bit mask (a numeric constant) which is the last input. The next to last input is a string constant that is a list of characters corresponding to each possible state. The number of possible states is two to the number-of-check-items power. For example, if there were five total inputs, there would be three inputs besides the last two constant inputs, and there would be eight possible states (two to the third power), and thus eight characters in the string constant. For each check input in turn, whether it has any of the bits on in the mask is determined, and this yes/no answer is converted to a digit in a binary number, starting with the least significant bit. The resulting number is then used to index into the string constant to get a single character, which is the function result.

## A2.2) Conversion functions

Conversion functions take one non-constant input and convert it to an item with a specified base data type, but with the same indexing and dimensions. By default, the conversion will be done by casting among numeric types, and through standard C format conversion between string and numeric types. The conversion can also be done with a lookup table, assuming compatibility between the input major type, requested output major type, and the flavor of the lookup table. If a value is null in the input, it will become null in the output unless a lookup table has been designed to provide a non-null translation for a null input.

Function Name	Description
<code>string</code>	Converts any arbitrary input to a string item. This conversion function can take parameters. The first parameter is the length of the strings in the output item and the second is the format to use for data conversion. Along with these two parameters, one may optionally supply a string to use for null input data (an empty string is default), a multiplier, and an offset. Alternatively, if the first parameter is <i>green</i> , <i>Z</i> , or <i>legend</i> , a string representing a time will be output, assuming the input is a numeric

1/1/70 based UNIX time. This is most easily obtained from an item with a `netcdf\_id` of `_times_`. A numeric value of 0 will result in the current time being formatted, and a null will result in an empty string. Finally, if a single parameter *recursive* is provided along with a string to string lookup table, that table will be used in a way that attempts to translate all substrings in the input string instead of just the whole string. If parameter values are not provided, meaningful defaults will be used. Other than the case of the *recursive* parameter, parameters are meaningless if a usable lookup table is provided for the item.

float	Converts any arbitrary input to a float item. This conversion function can optionally take one parameter, <i>encode</i> . This is applicable only when the input argument is of type string. In this case, it will treat up to the first three characters in the string as a base 256 integer, then that number will be converted to a base 10 floating point value. This allows small text strings to be sent to the volume browser in floating point values.
int	Converts any arbitrary input to an int item. This conversion function can optionally take one parameter, either <i>order</i> or <i>count</i> . For both of these, the result is just the position in the output arrays, starting at zero. For a non-list item it is the station/record index, and for a list item it is list position. <i>count</i> is sensitive to whether items are null, whereas <i>order</i> is not.
short	Converts any arbitrary input to a short item.

### A2.3) Mathematical functions

Mathematical functions will operate on any numeric type as input, but not on strings. When base data types are mixed, the output type will be whichever of the input types has the greatest range of values. With the exception of `accum` and `mean`, if any of the inputs for a given station/record and list index (if applicable) is null, then the result will be null.

Function Name	Description
diff	Two inputs; result is the first input minus the second.
div	Two inputs; result is the first input divided by the second.
rem	Two inputs; result is the remainder from dividing the first input by the second.
add	Normally two or more inputs; result is the sum of all inputs. If there is just one list input, then the result is a scalar and each item in the list is summed to get the scalar result. If there is one scalar input, then the sum is done in record/station space and copied to each item in the output.
accum	Normally two or more inputs; result is the sum of all non-null inputs. If there is just one list input, then the result is a scalar and each item in the list is summed to get the scalar result. If there is one scalar input, then the sum is done in record/station space and copied to each item in the output. This differs from `add`, where the presence of any null input will result in a null output. However, if all inputs are null, the result will

	still be null.
mult	Normally two or more inputs; result is the product of all inputs. If there is just one list input, then the result is a scalar and each item in the list is multiplied to get the scalar result. If there is one scalar input, then the multiplication is done in record/station space and copied to each item in the output.
avg	Normally two or more inputs; result is the average of all inputs. If there is just one list input, then the result is a scalar and each item in the list is summed to get the scalar result. If there is one scalar input, then the averaging is done in record/station space and copied to each item in the output.
mean	Normally two or more inputs; result is the average of all non-null inputs. If there is just one list input, then the result is a scalar and each item in the list is summed to get the scalar result. If there is one scalar input, then the averaging is done in record/station space and copied to each item in the output. This differs from `avg,' where the presence of any null input will result in a null output. However, if all inputs are null, the result will still be null.
lintrans	Two or more inputs. If I1, I2, etc., are the inputs and R is the result, then $R = I1 * I2 + I3 * I4...$ Using standard mathematical precedence, all the multiplications take place before the additions. If there is an odd number of inputs, the last is simply added to the result without being multiplied by anything.

#### A2.4) Meteorological functions

Meteorological functions operate only on float or numeric constant item types; any null inputs will result in a null output.

Function Name	Description
ucomp	Two inputs, a magnitude and a direction; result is the u component of that vector.
vcomp	Two inputs, a magnitude and a direction; result is the v component of that vector.
dir	Two inputs, a u component and a v component; result is the compass direction from of that vector.
mag	Two inputs, a u component and a v component; result is the magnitude of that vector.
heat_index	Two inputs, temperature and dewpoint in kelvins; result is the Heat Index in kelvins.
wind_chill	Two inputs, temperature in kelvins and wind speed in meters per second; result is the Wind Chill temperature in kelvins.
dewpoint	Two inputs, temperature in kelvins and relative humidity in percent. Alternatively, three inputs, pressure in millibars, temperature in kelvins, and specific humidity in g/kg. Result is the dewpoint in kelvins.
temperature	Two inputs, potential temperature in kelvins and pressure in millibars; result is the

	temperature in kelvins.
theta	Two inputs, temperature in kelvins and pressure in millibars; result is the potential temperature in kelvins.
spechum	Two inputs, pressure in millibars and dewpoint in kelvins. Alternatively, three inputs, pressure in millibars, temperature in kelvins, and relative humidity in percent. Result is the specific humidity in g/kg.
thetae	Three inputs, pressure in millibars, temperature in kelvins and dewpoint in kelvins; result is the equivalent potential temperature in kelvins.
alt2press	Two inputs, altimeter setting in millibars and elevation in meters; result is the surface station pressure in millibars.
height_of	One input, a numeric constant which is a pressure in millibars; result is a scalar station indexed item of float base type, which contains the corresponding height of that pressure surface for each station, interpolated from gridded data. The user may supply via a 'parameters' keyword a list of gridded data source names to try; otherwise a default list will be used. If gridded data access fails, the result will be based on a standard atmosphere.
ztopsa	Returns a pressure in millibars based on a height in meters using a standard atmosphere.
ptozsa	Returns a height in meters based on a pressure in millibars using a standard atmosphere.
elev	No arguments; result is a scalar station indexed item of float base type, which contains the elevation of each station as obtained from static metadata. (The existence of this function does not prevent one from reading the elevation from a netCDF variable if such a variable exists in the data set.)

## A2.5) Logical functions

Logical functions have the same output base type as the first argument, but the maximum dimensionality of all the arguments. With a few exceptions, they will take any mix of base types as input. In general, logical functions test the truth of some relationship among the inputs. If true, the first non-null value among the inputs is placed in the output; if false, the output is a null value. Where the first non-null value is string and the output is numeric, the value 0 will be placed in the output. Where the first non-null value is numeric and the output is string, the string "0" will be placed in the output.

Function Name	Description
or	Two or more inputs; result is the first non-null value among the inputs. 'or' is different from all other functions in that it will work even if some of its inputs are not definable because of mismatches between assumed and actual netCDF variable names.

and	Two or more inputs; if all inputs are non-null, then the value from the first input will be placed in the output.
nor	Two inputs. If both inputs are null, then a non-null (0 for numeric, "0" for string) will be placed in the output.
xor	Two inputs. If one input is null and one input is non-null, then the value of the non-null input will be placed in the output.
not	One input. If the input is null, then a non-null (0 for numeric, "0" for string) will be placed in the output.
==	Two inputs, which may not mix string and numeric types. If the two inputs test as equivalent, then the first input is copied to the output.
!=	Two inputs, which may not mix string and numeric types. If the two inputs test as not equivalent, then the first input is copied to the output.
>	Two inputs, which may not mix string and numeric types. If the first input tests as greater than the second input, then the first input is copied to the output.
<	Two inputs, which may not mix string and numeric types. If the first input tests as less than the second input, then the first input is copied to the output.
>=	Two inputs, which may not mix string and numeric types. If the first input tests as greater than or equal to the second input, then the first input is copied to the output.
<=	Two inputs, which may not mix string and numeric types. If the first input tests as less than or equal to the second input, then the first input is copied to the output.
min	Normally two or more inputs, which may not mix string and numeric types. If any values are non-null, then the first non-null value that tests as being less than or equal to all the others is copied to the output. If there is just one list input, then the result is a scalar and the minimum non-null of the list gets copied to the output. If a single scalar, then the minimum non-null of all the records/stations gets copied to each record of the output.
max	Normally two or more inputs, which may not mix string and numeric types. If any values are non-null, then the first non-null value that tests as being greater than or equal to all the others is copied to the output. If there is just one list input, then the result is a scalar and the maximum non-null of the list gets copied to the output. If a single scalar, then the maximum non-null of all the records/stations gets copied to each record of the output.
bit_none	Two inputs, which must have a short or int base type or be a numeric constant. If there are no bits turned on in both inputs, then a value of zero will be placed in the output, otherwise a null output will result.
bit_any	Two inputs, which must have a short or int base type or be a numeric constant. If there are any bits turned on in both inputs, then a value of zero will be placed in the output, otherwise a null output will result.
bit_all	Two inputs, which must have a short or int base type or be a numeric constant. If every bit that is on in the first input is also on in the second output, then a value of

zero will be placed in the output, otherwise a null output will result.

`none_there` Two or more string inputs. If none of the remaining inputs appears in the first input as a substring, then the first input will be placed in the output, otherwise a null output will result.

`any_there` Two or more string inputs. If any of the remaining inputs appears in the first input as a substring, then the first input will be placed in the output, otherwise a null output will result.

`all_there` Two or more string inputs. If all of the remaining inputs appear in the first input as a substring, then the first input will be placed in the output, otherwise a null output will result.

## A2.6) List handling functions

Function Name	Description
<code>down_interp</code>	Performs vertical interpolation in the case where the value of the vertical coordinate being used increases downward. This function takes three inputs. The first, a float list, is the vertical coordinate list; the second, also a float list, is the list of values to interpolate; and the third, a float list, float scalar, or numeric constant, is the specific vertical coordinate value(s) to which to interpolate. This is one of only six functions that can take list items of different lengths, though the first two lists must have the same length. The output is a float whose dimensionality is determined only by that of the final argument, as opposed to the usual behavior of using the maximum dimensionality of all arguments. By default, it is possible to interpolate data from within a vertical gap 2 levels wide (one missing level). The size of this gap can be specified optionally as an input to a <code>`parameters'</code> keyword. Furthermore, if two parameters are supplied, the second parameter is the largest vertical gap in terms of the vertical coordinate over which interpolation is allowed.
<code>up_interp</code>	Performs vertical interpolation in the case where the value of the vertical coordinate being used increases upward. This function takes three inputs. The first, a float list, is the vertical coordinate list; the second, also a float list, is the list of values to interpolate; and the third, a float list, float scalar, or numeric constant, is the specific vertical coordinate value(s) to which to interpolate. This is one of only six functions that can take list items of different lengths, though the first two lists must have the same length. The output is a float whose dimensionality is determined only by that of the final argument, as opposed to the usual behavior of using the maximum dimensionality of all arguments. By default, it is possible to interpolate data from within a vertical gap 2 levels wide (one missing level). The size of this gap can be specified optionally as an input to a <code>`parameters'</code> keyword. Furthermore, if two parameters are supplied, the second parameter is the largest vertical gap in terms of the vertical coordinate

over which interpolation is allowed.

`down_sample` Performs vertical sampling in the case where the value of the vertical coordinate being used increases downward. Sampling means directly using the value from the nearest level rather than interpolating. This function takes three inputs. The first, a float list, is the vertical coordinate list; the second, also a float list, is the list of values to sample from; and the third, a float list, float scalar, or numeric constant, is the specific vertical coordinate value(s) to sample for. This is one of only six functions that can take list items of different lengths, though the first two lists must have the same length. The output is a float whose dimensionality is determined only by that of the final argument, as opposed to the usual behavior of using the maximum dimensionality of all arguments. For sampling, the default behavior is not to allow sampling from within vertical gaps of missing data, and not to allow the same item in the input list to be used more than once (oversampling) in the case of multiple output levels. One can change this behavior with a parameter. The absolute value of the parameter is the size of a gap from which to allow sampling (a gap of 2 means allow one missing level), and if the parameter is positive, the oversampling is allowed. Furthermore, if two parameters are supplied, the second parameter is the largest vertical gap in terms of the vertical coordinate from within which sampling is allowed.

`up_sample` Performs vertical sampling in the case where the value of the vertical coordinate being used increases upward. Sampling means directly using the value from the nearest level rather than interpolating. This function takes three inputs. The first, a float list, is the vertical coordinate list; the second, also a float list, is the list of values to sample from; and the third, a float list, float scalar, or numeric constant, is the specific vertical coordinate value(s) to sample for. This is one of only six functions that can take list items of different lengths, though the first two lists must have the same length. The output is a float whose dimensionality is determined only by that of the final argument, as opposed to the usual behavior of using the maximum dimensionality of all arguments. For sampling, the default behavior is not to allow sampling from within vertical gaps of missing data, and not to allow the same item in the input list to be used more than once (oversampling) in the case of multiple output levels. One can change this behavior with a parameter. The absolute value of the parameter is the size of a gap from which to allow sampling (a gap of 2 means allow one missing level), and if the parameter is positive, the oversampling is allowed. Furthermore, if two parameters are supplied, the second parameter is the largest vertical gap in terms of the vertical coordinate from within which sampling is allowed.

`find_first` One input, which can be any kind of list. The result is a scalar int item, which contains the list index of the first non-null item in the list.

`find_last` One input, which can be any kind of list. The result is a scalar int item, which contains the list index of the last non-null item in the list.

`find_max` One input, which can be any kind of list. The result is a scalar int item, which contains the list index of the first non-null item in the list that tests greater than

or equal to all the others.

`find_min` One input, which can be any kind of list. The result is a scalar int item, which contains the list index of the first non-null item in the list that tests less than or equal to all the others.

`index` Two inputs, the first of which is any kind of list. The second must be either a numeric constant or a scalar int, most often from one of the functions `'find_first,' 'find_last,' 'find_max,'` or `'find_min.'` The value in the first input whose list index corresponds to the second input is placed in the output item, which is a scalar of the same base type as the input.

`select` One input, any kind of list. A lookup table must be supplied using `'function_table.'` The table used must have numeric output, and its input must be compatible with the major type of the list. The table is used to arbitrarily rank each item in the list. Whichever item has the lowest rank number is selected and placed in the output item, which is a scalar of the same base type as the input.

`catenate` One or more string inputs of any dimensionality. This is one of only six functions that can take list items of different lengths. Each individual string, whether it is part of a list or scalar, is catenated to the eventual result string. The output item is a scalar string type. The default delimiter between each string is an empty string. One can optionally supply a different delimiter as a single argument to the `'parameters'` keyword. If one supplies two arguments to the `'parameters'` keyword, these are treated as a leading and trailing string. Finally three arguments are treated as a leading, delimiting, and trailing string.

`group` Two or more inputs that all have the same base type but any dimensionality. This is one of only six functions that can take list items of different lengths. Each individual input value, whether it is part of a list or scalar, is added to the eventual resulting output list. The output item is a list of the same base type as the inputs. Optionally, if one supplies the parameter *like*, an output list object will be created with the same base type and record/station type as the first input, but having a number of items equal to the number of additional parameters after *like*. These additional parameters will be used to provide values for the items in each list, creating essentially a 'list constant'.

`sort_up` This function takes as input one or two list arguments. The basic purpose of this function is to sort a list increasing upward. If one argument, sorts items in the list increasing upward, nulls last. If two arguments, then the first list is sorted according to the values in the second list.

`sort_dn` This function takes as input one or two list arguments. The basic purpose of this function is to sort a list increasing downward. If one argument, sorts items in the list increasing downward, nulls last. If two arguments, then the first list is sorted according to the values in the second list.

`list_filter` This function takes as input one or two list arguments. The basic purpose of this function is to extract nulls from a list. This function optionally takes one parameter, which limits the physical size of the output list, which is otherwise

the same size as the input lists. If one argument, packs all non-null items to the front of the list. If two arguments, then each time a non-null is encountered in the second list, the corresponding item in the first is copied to the next position in the output. In either case if there are not enough non-nulls to fill the output list, it is padded to the end with nulls.

`list_limit` This function takes as input one list argument followed by up to 3 optional scalar numeric arguments. This function optionally takes one parameter, which limits the physical size of the output list, which is otherwise the same size as the input list. The basic purpose of this function is to create a list that is a subset of the input list. The first optional argument is the number of items to copy to the output list, which defaults to its physical size. The second is the index of the input list from which to start copying, which defaults to zero. The third is the last item in the input list to copy from, and this defaults to just enough to allow consecutive copying. If the range of items to copy from is larger than the number of items to copy to the output list, the first and last items requested will still be copied, but items will be skipped within the input range to cover it. If there are not enough copyable items to fill the output list, it is padded to the end with nulls.

`make_layer` This function takes from two to five float items or numeric constants. Any of the first three may be list items. This function is used to construct a 'layer' item, which is a specialized implementation of a float list item. A 'layer' always has an item count of a multiple of three. Each triplet consists of a minimum value for the layer, a representative value for the layer, and a maximum value for the layer. If list arguments are supplied, then one ends up with a list containing three times the number of items in the list, representing multiple layers in a single item. If two arguments are supplied, these are the min and max layer values. If three arguments are supplied, these are the min, representative, and max layer values. The fourth and fifth arguments are an optional multiplier and offset, which allows one to do a units conversion when constructing the layer. It is meaningful to supply nulls for some of the three arguments.

`layer_overlap` This function takes one or two float list arguments that represent layers, and outputs a layer argument. If there is one list argument that contains that contains one layer, then any nulls that can possibly be assigned values based on other non-nulls in the layer will be defined. One argument that contains multiple layers will result in computing the overlap (intersection) among all layers in the list. Two arguments will result in computing the overlap between the two arguments.

`layer_union` This function takes one or two float list arguments that represent layers, and outputs a layer argument. If there is one list argument that contains one layer, then any nulls that can possibly be assigned values based on other non-nulls in the layer will be defined. One argument that contains multiple layers will result in computing the union among all layers in the list. Two arguments will result in computing the union between the two arguments.

layer_min	This function takes one float list input that is a layer and one optional second int scalar or numeric constant. The second input is meaningful only when the first input contains multiple layers (the number of items is 6, 9, etc.), and allows one to refer to a specific layer (0th, 1st, etc.). The output of this function is a float scalar containing the minimum value of the selected layer, which defaults to the 0th if no second argument is present.
layer_rep	This function takes one float list input that is a layer and one optional second int scalar or numeric constant. The second input is meaningful only when the first input contains multiple layers (the number of items is 6, 9, etc.), and allows one to refer to a specific layer (0th, 1st, etc.). The output of this function is a float scalar containing the representative value of the selected layer, which defaults to the 0th if no second argument is present.
layer_max	This function takes one float list input that is a layer and one optional second int scalar or numeric constant. The second input is meaningful only when the first input contains multiple layers (the number of items is 6, 9, etc.), and allows one to refer to a specific layer (0th, 1st, etc.). The output of this function is a float scalar containing the maximum value of the selected layer, which defaults to the 0th if no second argument is present.

### Appendix 3) Display Methods

Method	Description
formatted	Normally one scalar input of any base type. This method plots a data value using a C format specifier provided in the <code>`format'</code> keyword. The argument to the <code>`format'</code> keyword can also be the <code>item_id</code> of a string constant from which to obtain the format, and this is indicated by the leading double dollar sign ( <code>\$\$</code> ). After a string has been produced with the specified format, this can also be passed through complex formatting if an argument was supplied to the <code>`complex_fmt'</code> keyword. The positioning of the plotted element responds to the nine standard position options available as an argument to the <code>`placement'</code> keyword, or to the coordinates supplied as arguments to <code>`location'</code> if the placement is <i>free</i> . If numeric, the value undergoes a units conversion based on the arguments to <code>`multiplier'</code> and <code>`offset,'</code> and will be plottable only if the converted value is within the bounds set by the arguments to <code>`min_trans'</code> and <code>`max_trans.'</code> If the value is null or otherwise deemed unplotable, the string supplied as an argument to <code>`undef_string'</code> can be plotted instead of nothing. A character set other than ASCII can be used to plot the resulting string based on the argument to <code>`alt_char_set.'</code> It is possible to strip a specified number of characters off of the front of the formatted string before plotting by supplying an argument to the keyword <code>`trim_count.'</code> The size of the item plotted can be made different from the default size by supplying an argument to the keyword <code>`magnification.'</code> If the item being displayed is numeric, the format can be determined by the output of a string to number lookup table provided the file name of the table is supplied as an argument to the <code>`table_file'</code> keyword.

There is a new feature that allows one to plot text at an arbitrary location on the frame using this method, and this is triggered by having more than one input, or by having a constant input. The plain language positional arguments to 'placement' work the same, except relative to the whole display frame. If *free* is used, the arguments to 'location' must be a frame position, (0,0) being lower left and (1,1) being upper right. If a constant argument is given, then that one data item is formatted and placed at the indicated position in the frame. If multiple inputs are given, then the second and/or third inputs are assumed to be numeric constants that refer to a station/record index or a range of these indices. In this case, each datum in the range of indices is formatted and catenated to produce a single string that is written to the indicated position in the frame. If an argument was given to the 'complex\_fmt' keyword, this will be applied once after this catenation operation.

lookup One scalar input of any base type. This display method plots the result of a simple lookup table translation. The lookup table, the file for which is supplied as an argument to the 'table\_file' keyword, needs to have string output and input compatible with the major type of the item being plotted. The positioning of the plotted element responds to the nine standard position options available as an argument to the 'placement' keyword, or to the coordinates supplied as arguments to 'location' if the placement is *free*. An alternate character set can be selected using 'alt\_char\_set,' and the plotted size of the item can be modified by supplying a 'magnification.'

translation One scalar string input. The output string is generated by performing a recursive translation using a string to string lookup table, supplied as an argument to the 'table\_file' keyword. Any text that is successfully translated will be displayed by whatever character set is specified as an argument to 'alt\_char\_set.' Untranslated text will be shown in plain ASCII. The positioning of the plotted element responds to the nine standard position options available as an argument to the 'placement' keyword, or to the coordinates supplied as arguments to 'location' if the placement is *free*. The plotted size of the item can be modified by supplying a 'magnification'.

trend This is a specialized method used to display the standard coding for the three hour pressure tendency. Inputs are the change amount (float scalar) and the pressure change character (short scalar). Any desired units conversion for the change amount must be supplied as an argument to 'multiplier,' but 'offset' is ignored. The formatting of the change amount can also be set via 'format.' The positioning of the plotted element responds to the nine standard position options available as an argument to the 'placement' keyword, or to the coordinates supplied as arguments to 'location' if the placement is *free*. The plotted size of the item can be modified by supplying a 'magnification.'

barb This method displays vector data as wind barbs. It takes two to four inputs, which must all be scalar floats. These are wind speed and direction, and optionally gust speed and direction. If gust speed is supplied without direction, the direction will be taken to be the same as input two. When gust information is available, it is plotted

as an additional arrow with the gust speed labeled in ASCII. The position is predetermined by the method, so *free* 'placement' is required. 'max\_trans' can be used to specify a maximum valid wind speed. Speeds are assumed in knots; 'multiplier' can be used for units conversion. 'min\_trans' specifies the smallest gust speed that will be plotted as such, and 'offset' states the minimum vector difference that must exist between the wind and the gust before a gust will be plotted. The argument to 'magnification' is overloaded: its magnitude is the size relative to the default of the barb to be plotted, while if it is negative, "calm" circles will not be plotted. If the barb as a whole is deemed unplotable, normally nothing will be plotted, but 'undef\_string' can be used to specify some ASCII string to plot in that case.

arrow

This method displays vector data as arrows. The two scalar float inputs represent wind speed and direction. The length of the arrow is fixed, and the speed is plotted in ASCII at the end of the arrow. The position is predetermined by the method, so one must specify *free* 'placement.' If supplied, 'max\_trans' specifies the maximum wind speed considered valid. Speed units conversion is accomplished through 'multiplier,' but 'offset' is not used. Relative arrow size is specified using 'magnification.' If the arrow as a whole is deemed unplotable, normally nothing will be plotted; 'undef\_string' can be used to specify an ASCII string to plot in that case.

barbuv

Like 'barb,' this method displays vector data as wind barbs. The inputs differ however. Two to four float inputs are accepted. If there are only two inputs they are scalars representing u and v components of the wind, and a single barb is plotted. For three or four inputs, all are lists, representing u component, v component, height, and QC parameters, respectively. If there are three inputs, then a stack of wind barbs is plotted. If four, the wind barbs in the stack are colored according to the speed and the QC information. *free* 'placement' is required, since the position is predetermined by the method. 'max\_trans' can be used to specify a maximum plotable wind speed. Speeds are assumed in knots; 'multiplier' can be used for units conversion, but no 'offset' is applied. The argument to 'magnification' is overloaded: its magnitude is the relative size of the plotted barb, while if it is negative, "calm" circles will not be plotted. If the barb as a whole is deemed unplotable, normally nothing will be plotted, but 'undef\_string' can be used to specify an ASCII string to plot in that case.

arrowuv

Like 'arrow,' this method displays vector data as arrows. Two to four float inputs are accepted. If there are only two inputs they are scalars representing u and v components of the wind, and a single arrow is plotted. For three or four inputs, all are lists, representing u component, v component, height, and QC parameters, respectively. If there are three inputs, then a stack of arrows is plotted. If four, the arrows in the stack are colored according to the speed and the QC information. *free* 'placement' is required, since the position is predetermined by the method. 'max\_trans' can be used to specify a maximum plotable wind speed. Speed unit conversion is accomplished through 'multiplier,' but no 'offset' is applied. The argument to 'magnification' is overloaded: its magnitude is the relative size of the

plotted arrow, while if it is negative, "calm" circles will not be plotted. If the arrow as a whole is deemed unplottable, normally nothing will be plotted; ``undef_string'` can be used to specify an ASCII string to plot in that case.

circle This method allows one to draw a circle of arbitrary size with various visual characteristics. It takes at a minimum one numeric scalar input, which is the radius of the circle. This is multiplied by the value of ``multiplier'` to get an intermediate radius value, but no ``offset'` value is applied. If the intermediate radius value is less than zero the final pixel radius value is the absolute value, otherwise the intermediate value is assumed to be in kilometers, and is converted to get the proper final radius in pixels dependent on the current scale and zoom. If the final pixel radius is outside the range provided by arguments to ``min_trans'` and ``max_trans,'` it will not be plotted. After this check, radii supplied in pixels will be subject to the screen magnification factor, but radii supplied in kilometers will not. If the resulting radius is smaller than the value of the ``offset'` keyword, then that offset value will become the radius used. If an optional second non-list numeric argument is supplied, that will be used for the thickness of the circle, which will otherwise default to the argument to the keyword ``magnification,'` which defaults to one. If an optional third non-list numeric argument is supplied, that will be used for the circle attributes, which will otherwise default to the argument to the keyword ``attributes,'` which defaults to zero. The circle attributes are a bit mask, with the 1 bit on meaning a broken circle is drawn, the 2 bit on meaning a densely spiked circle is drawn, and the 4 bit on meaning a sparsely spiked circle is drawn.

polyline This method allows one to draw an arbitrary polyline. At a minimum, the polyline takes two float list arguments. If the placement is *free*, then these are assumed to be latitude and longitude, otherwise these are assumed to be x and y pixel displacements from the location of the data, which are subject to magnification. An optional third non-list string argument is a character to plot at the vertices, which will be modified by the ``alt_char_set'` keyword argument. The argument to the ``attributes'` keyword can be used to modify how the data location is treated in drawing the polyline. An ``attributes'` value of zero (default) will do nothing with the data location. Less than zero means attach the data location to the beginning of the polyline, greater than zero means attach it to the end. If the absolute value of the ``attributes'` argument is greater than one, then the vertex string will be plotted at the data location as well.

#### **Appendix 4) Constants with predefined meanings**

`id_to_use:`

This string constant allows one to temporarily define a single station whose data to access in a plan view design file. This can either be the ID of the station or `***` followed by a file name where the station ID can be found.

#### FRAME\_TIMES:

This string constant, if set to *TRUE*, designates that the inventory for this product should be the set of existing frame times if loaded as an overlay.

#### REFRESH:

This string constant, if set to *TRUE*, designates that all data items in the design file should be reread each time there is a zoom or pan, whether any items have the 'recompute' keyword defined or not.

#### smoothing\_distance:

This numeric constant applies if one is using the design file to gather data for an on-the-fly analysis. This is the scale distance of the smoothing in kilometers.

#### DENSITY:

This numeric constant, if present, acquires the current value of the screen density. Usually useful only if REFRESH is *TRUE* or some function items are designated with *recompute true*.

#### MAGNIFICATION:

This numeric constant, if present, acquires the current value of the screen magnification. Usually useful only if REFRESH is *TRUE* or some function items are designated with *recompute true*.

#### PROG\_DISC\_PAR:

This numeric constant, if present, acquires the current value of the progressive disclosure threshold, which is in units of pixels per kilometer. Usually useful only if REFRESH is *TRUE* or some function items are designated with *recompute true*.

#### EXTRA\_LEGEND:

This string constant, if present, is a string to pass out as additional information to add to the product legend. This string can be created using functions that result in a string constant output. The format of the string is *sss|iii*, where *sss* is a search string of any length, and *iii* is the string to insert into the legend, again of any length, and the vertical bar is literal. If no vertical bar is present, the whole string is inserted at the front of the legend string, and if the search string is zero length, what follows the vertical bar is appended to end of the legend string. Otherwise, what follows the vertical bar is inserted at the first occurrence of the search string. Multiple vertical bars in succession result in searching for additional occurrences of the search string. For example, if the argument defining the value of this string constant were *~||FOO~*,

this would result in the string FOO followed by a space being inserted at the second space in the legend, because as the reader will recall the default parsing of design files means that ~ is interpreted as a space.

## Appendix 5) Enhancements to the netCDF files

Before one attempts to manipulate the display characteristics of point data sets, it is helpful to understand the enhancements that have been made to the netCDF files to support this. Here, in CDL format, are the variables and attributes that have been added:

dimensions:

```
maxStaticIds = int;  
totalIdLen = int;  
nInventoryBins = int;
```

variables:

```
int  
    nStaticIds;  
    nStaticIds:_FillValue = 0;  
char  
    staticIds(maxStaticIds, totalIdLen);  
    staticIds:_FillValue = '\0';  
int  
    lastRecord(maxStaticIds);  
    lastRecord:_FillValue = -1;  
long  
    invTime(record);  
    invTime:_FillValue = 0;  
int  
    prevRecord(record);  
    prevRecord:_FillValue = -1;  
long  
    inventory(maxStaticIds);  
    inventory:_FillValue = 0;  
long  
    globalInventory;  
    globalInventory:_FillValue = 0;  
int  
    firstOverflow;  
    firstOverflow:_FillValue = -1;  
int  
    isOverflow(record);  
    isOverflow:_FillValue = 0;  
int
```

```

        firstInBin(nInventoryBins);
        firstInBin:_FillValue = -1;
    int
        lastInBin(nInventoryBins);
        lastInBin:_FillValue = -1;

// global attributes:
:cdlDate = "ascii_date";
:idVariables = "netcdf_varname1,netcdf_varname2...";
:timeVariables = "netcdf_varname1,netcdf_varname2...";
:filePeriod = int;
:fileEndOffset = int;
// optional global attributes
// :latLonVars = "latitude_var,longitude_var";
// :log_rewrites = "true";
// :have_forecasts = "true";
// :forecast_list = "fcst_time1,fcst_time2,...";
// :max_open = int;
// :cache_metadata = "false";

```

The main reason to be familiar with these enhancements is so that in the event that one needs to add an additional variable or attribute to an existing netCDF file, one can avoid using these reserved variables and attributes. We refer to these enhancements as 'record management' variables and attributes. The primary reason these record management variables and attributes exist is to optimize performance for inventory and for sequential access of data for a single station. An implementor also needs to understand that the record management variables are not displayable.

Of all the new global attributes, the :cdlDate attribute is the most important. If one ever changes the structure of the data files in a directory (add, delete, or move a variable) the value of the :cdlDate attribute needs to be changed. This allows the software to recognize that it has to treat the new files differently.

At times it may be important to recognize that the data set has a forecast component. If this is the case, then the optional global attribute :have\_forecasts must be present. If its value is simply 'true', then individual records are identified with both a valid and forecast time and files will be time stamped with the reference (initial) time of the data. If its value is the name of a dimension, then that dimension is assumed to represent individual forecast times, and it is assumed that all client variables will have that dimension in them. The specific forecast times along that dimension must be enumerated in the forecast\_list global attribute, and individual records are identified with the reference time of the data.

Also, certain data sets are identified by latitude and longitude instead of an ASCII station ID. If this is the case, then the :idVariables attribute will list the latitude and longitude variables.

If one is trying to create a new data set, it is probably best to start with one of the existing default .cdl files that is closest to the new data set and modify it to accommodate your client variables, rather than trying to replicate this list of record management variables.

## Appendix 6) Standard lookup tables

Several lookup tables are included with the AWIPS software. They are listed here by type, followed by an example of each for reference.

Type	Files
	cloud_chars.txt
	cloud_chars_nom.txt
s2s	qc_check_bad.txt
	qc_check_fmt.txt
	qc_check_good.txt
	wx_symbol_trans.txt
s2n	cloud_select.txt
	rank_report_type.txt
	fractions_lookup.txt
	ldad_prcp_formats.txt
n2s	maritime_cloud_chars.txt
	prcp_formats.txt
	raob_dd_char.txt
n2n	prcp_goodness.txt

### wx\_symbol\_trans.txt

```
s2s
right VC
-SHRA : 45 54
+SHRA : 180 54
SHRA  : 79 54
-SHSN : 46 54
+SHSN : 179 54
SHSN  : 89 54
-TSRA : 45 41
+TSRA : 79 88
TSRA  : 180 41
-TSSN : 46 41
+TSSN : 179 88
TSSN  : 89 41
FZRASN : 71 89
BCBR   : 35
MIBR   : 36
BR     : 34
FZFG   : 173
```

BCFG : 63  
MIFG : 64  
FG : 53  
TS : 41  
+TS : 88  
FC : 43  
+FC : 137  
PO : 32  
DRSN : 59  
+DRSN : 60  
BLSN : 61  
+BLSN : 62  
FU : 28  
HZ : 29  
-SH : 54  
SH : 78  
DU : 30  
SA : 31  
SS : 55  
DS : 55  
+SS : 58  
+DS : 58  
-FZRA : 70  
-FZDZ : 68  
FZRA : 71  
FZDZ : 69  
GR : 80  
IC : 76  
PE : 77  
PL : 77  
GS : 75  
DZRA : 73  
RADZ : 73  
RASN : 47  
SNRA : 47  
-RA : 45  
+RA : 180  
RA : 79  
-DZ : 44  
+DZ : 181  
DZ : 67  
-SN : 46  
+SN : 179  
SN : 89  
-UP : 174  
+UP : 176  
UP : 175  
IP : 77  
SG : 75  
VA : 177  
PRFG : 178  
SQ : 42  
BLDU : 31

(The output weather string represents one or more [weather symbols](#).)

### **rank\_report\_type.txt**

```
s2n  
METAR 1  
SPECI 2
```

(The standard MTR plots prefer a METAR over a SPECI if both are available in a given hour.)

### **prcp\_formats.txt**

```
n2s  
-0.005 0.005 ~T  
0.005 0.015 ~.o1  
0.015 0.025 ~.o2  
0.025 0.035 ~.o3  
0.035 0.045 ~.o4  
0.045 0.055 ~.o5  
0.055 0.065 ~.o6  
0.065 0.075 ~.o7  
0.075 0.085 ~.o8  
0.085 0.095 ~.o9  
0.095 0.495 %4.2f  
0.495 9.995 %5.2f  
9.995 100 %6.2f
```

(The "o" is used instead of "0" arbitrarily to distinguish low-precip reports. C-style formats are used for higher amounts, with a break at .50" to emphasize heavier precip.)

### **prcp\_goodness.txt**

```
n2n  
0 1 0 100000
```

(This allows higher precipitation amounts to be favored automatically by the progressive disclosure (declutter) mechanism.)

---

**Author: Jim Ramer**  
**Last update: 8 Oct 07**

## AWIPS character sets

There are currently five different AWIPS character sets. Each character set can conceivably have 255 printable characters in it, although currently none of them have that many implemented. When displaying character data in AWIPS, the implementor must select one of these character sets to use, and then all text display calls will use that character set until it is changed. AWIPS character fonts are stroked fonts rather than bitmapped fonts -- this is handy because they can be scaled arbitrarily rather than only by integral magnifications. The individual strokes are integer coordinates on a specific size matrix for each character set. Here is a list of these character sets and the *alt\_char\_set* identifiers used in design files:

<b>Dataset</b>	<b>Design file tag</b>
7x9 ASCII (default)	ascii
9x11 ASCII	large_ascii
weather symbols (11x11)	weather
special symbols (11x11)	special
large special symbols (16x20)	large_special

In practice, the overwhelming majority of display items use the default standard ASCII character set. Weather and special symbols (the latter for clouds) are used in several plots, and radar graphics use several of the special symbols (both sizes). The file `$FXA_HOME/data/metar_wx_symbols.dat` controls the translation of METAR weather strings by listing character sequences and the character code(s) in this character set to use. The volume browser displays that are fields of icons can in theory use any of these character sets, though currently none of these types of displays uses anything but standard ASCII or weather symbols.

Here are displays that show the individual fonts available in each character set. These are shown with a stretch factor of 2, but without the line width increase typical of AWIPS character magnification, so that the individual strokes can more easily be seen.

# Standard ASCII

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143

# Large ASCII

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

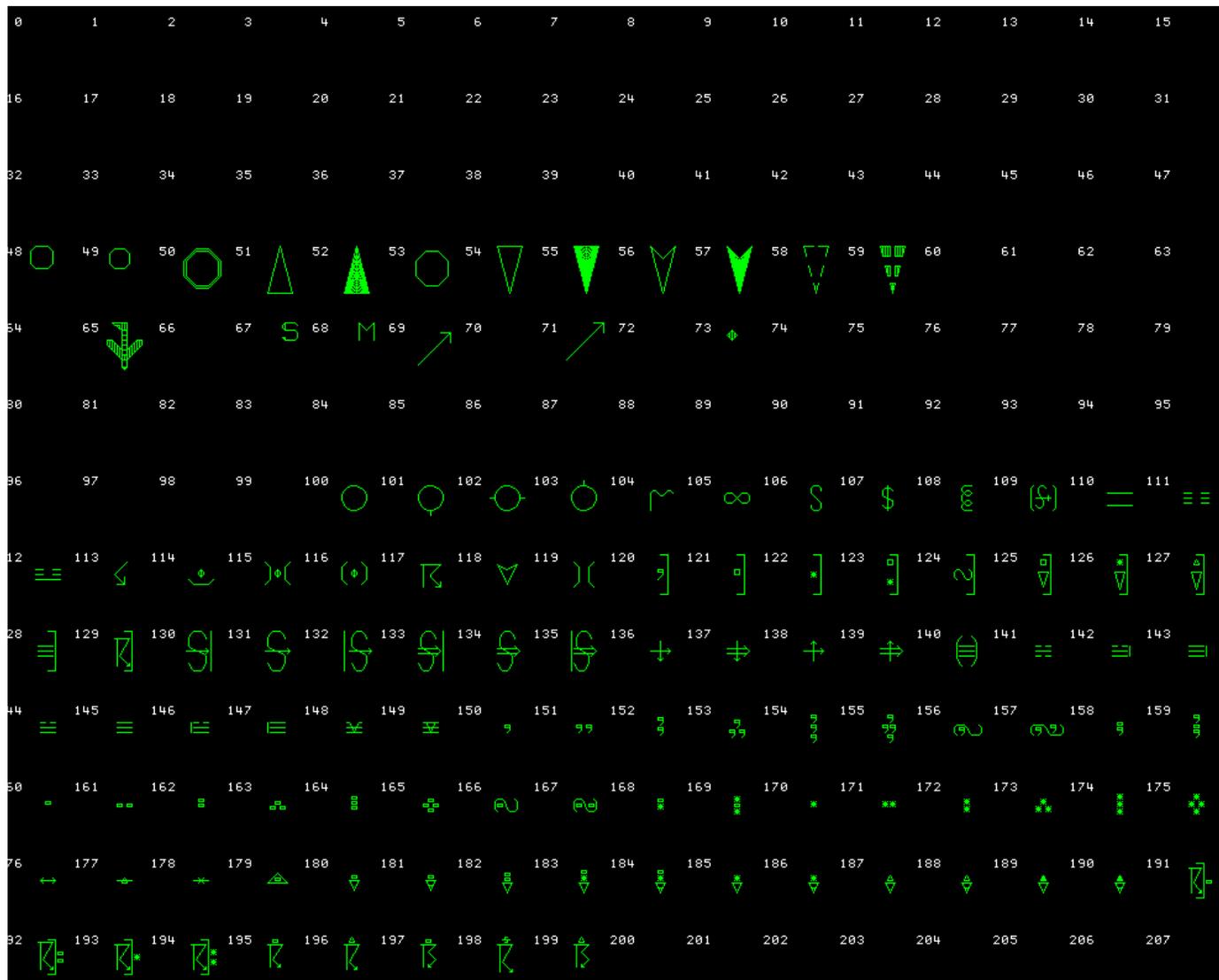
# Weather Symbols

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207

# Special Symbols

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79

# Large Special Symbols



Developers needing to design additional character displays can do so using the program `strokfont,' which is built in the fontData directory. In that same directory are the files that contain the stroke data for each character set. The files are:

set 0	standard ASCII	standard.bin
set 1	large ASCII	afosascii.bin
set 2	weather symbols	afosfonts.bin
set 3	special symbols	font2syms.bin
set 4	large special symbols	largefont.bin

**Author: Jim Ramer**  
**Last update: 3 Aug 01**

## Transition from pre-localization environment to the post-localization environment

After one has completely built the WFO advanced software tree, it is still not possible to run a workstation or an ingest system, because one must run a localization to create all of the site specific metadata. At one time, a complete software build in effect created a Denver localization, but this is no longer the case.

In order to use the localization feature, you first need to define the following environment variables (you might as well put these definitions in your login file):

```
setenv FXA_NATL_CONFIG_DATA ${FXA_HOME}/data/localization
setenv FXA_LOCALIZATION_ROOT ${FXA_HOME}/data/localizationDataSets
setenv FXA_LOCAL_SITE FSL
setenv FXA_INGEST_SITE FSL
setenv FXA_LOCALIZATION_SCRIPTS ${FXA_HOME}/data/localization/scripts
```

At a minimum, these are the utility programs that one needs to have built in order to create localizations...doing a full build including a buildexe in \$FXA\_HOME/src will assure that these are all built:

```
cd $FXA_HOME/src/dataMgmt
make fileMover keyMunge pasteUtil
cd $FXA_HOME/src/dm/grid
make makeGridKeyTables processStyleInfo testGridKeyServer initCdlTemplate
cd $FXA_HOME/src/dm/point
make reformatTest testPlotDesign
cd $FXA_HOME/src/dm/shapefile
make shp2bcd
cd $FXA_HOME/src/util
make textBufferTest
cd $FXA_HOME/src/geoLib
make bcdProc maksuparg test_grhi_remap maksubgrid rangeAzimuth
cd $FXA_HOME/src/geoTables
make GELTtest image_mask newGELTmaker
cd $FXA_HOME/src/mapping
make makthermo makxsect testDepictorTable
cd $FXA_HOME/src/staticPlotData
make va_driver masterToGoodness
```

In order to be able to also run all of the non-default tasks, one needs to build these as well

```
cd $FXA_HOME/src/tstorm/localize
make create_radarLoc sitefinder
$FXA_HOME/src/ffmp/localize
make localizeForFFMP
```

Once you have your tree built and have set the proper environment variables, you can create the default FSL localization by running the following commands:

```
cd ${FXA_HOME}/data/localization/scripts
```

mainScript.csh

The localization process will take 5-20 minutes depending on the type of machine you are running on and how busy it is...you will get some diagnostics along the way. When finished, just start your D2D as you normally do and it will be running an FSL localization, which is essentially a BOU localization with small modifications.

The output diagnostics from the localization process have been improved to the point where if you see something that looks like an error, it probably is. Because the localization stuff is mostly script driven, it is pretty quick to resynchronize yourself. Any time a problem does arise in creating a localization and you are sure you have all of your utility programs built, first run the following commands:

```
cd ${FXA_HOME}/src/localization
dv-sync -R .
make data
```

and then try to recreate the localization before assuming there is a bug somewhere. Remember that at some point before running a localization you must have done a `make data' for the whole software tree...it is not enough to just build the utility programs and to do a `make data' in \${FXA\_HOME}/src/localization.

The reason this creates an FSL localization is that the FXA\_LOCAL\_SITE and FXA\_INGEST\_SITE environment variables are both pointing to FSL. Whatever FXA\_LOCAL\_SITE points to, that is the localization your workstation will use.

It is just as easy to create additional localizations. If, for example, one issued the command:

```
mainScript.csh OUN
```

a Norman localization will be created on the assumption that the ingest machine you are looking at is running an FSL localization. This is because FXA\_INGEST\_SITE still points to FSL. If you want to create a fully Norman localization, run the command:

```
mainScript.csh OUN OUN
```

To then run a workstation using this Norman localization, set the environment variable FXA\_LOCAL\_SITE to OUN and start your D2D as you normally do and it will be running a Norman localization.

To get a list of available localizations, run the command:

```
${FXA_HOME}/data/localization/scripts/cwaIds.csh -a
```

---

**Author: Jim Ramer**  
**Last update: 29 Nov 04**

# Directives

For any WFO, it is possible to run a default localization to produce an operational configuration that is at least minimally usable. However, most sites will need at least some modifications to the default localization to produce the optimum operational configuration. The files installed in order to accomplish this are called *site-specific control files*.

Assume one is working with some arbitrary localization with the ID LLL. All of the site-specific control files for the LLL localization can be found in the directory `$FXA_HOME/data/localization/LLL`, and have file names that start with `LLL-`. (In a source tree, they can be found in `$FXA_HOME/src/localization/localData`.) In the site-specific control files called `LLL-mainConfig.txt` and `LLL-wwaConfig.txt` are software switches that can control many different aspects of how localization data sets are created. These switches are referred to as *directives*. `LLL-wwaConfig.txt` is a separate place for those switches that affect how `warnGen` behaves, and `LLL-mainConfig.txt` contains the rest.

Each directive is one line in the file, and looks like this:

```
@@@tag <the value of the directive>
```

The `@` signs are literal, the tag can be any string of alphanumeric characters with no white space, and the value of the directive can be any arbitrary text. By convention, tags are all caps. Most directives will acquire a default value if not present. If a directive occurs twice in a file, the last entry is used.

Additionally, it is possible to affect the value of directives by supplying customization files. These files reside in a directory pointed to by the environment variable `$FXA_CUSTOM_FILES`, or in `$FXA_CUSTOM_FILES/$FXA_CUSTOM_VERSION`. The customization main directive file can be named either `LLL-mainConfig.txt` or `mainConfig.txt`, and the customization `warnGen` directive file can be named either `LLL-wwaConfig.txt` or `wwaConfig.txt`. The ones with `LLL-` will affect only that localization, while the others will affect all localizations. By default, `FXA_CUSTOM_FILES` points to `$FXA_DATA/customFiles`. Since this is a cross-mounted disk, a change made here will affect future localizations run on any machine. See section 9 in [localization](#) for more details.

When one or more customization directive files are present, all like directive files are catenated in this order:

```
$FXA_HOME/data/localization/LLL/LLL-*  
$FXA_CUSTOM_FILES/*  
$FXA_CUSTOM_FILES/LLL-*  
$FXA_CUSTOM_FILES/$FXA_CUSTOM_VERSION/*  
$FXA_CUSTOM_FILES/$FXA_CUSTOM_VERSION/LLL-*
```

Because the last occurrence of a directive is used, entries in a directive file from a given directory in this list will override entries in files from a directory earlier in the list. This means, for

example, that customization directives override those from site-specific control files (found in localization/LLL).

In the case where a directive is being implemented as part of a centrally supplied software load, it needs to be implemented in a site-specific control file, whereas someone implementing a directive on site should use a customization file. This allows directives implemented on site to survive a new software load.

It is important to note that changes to directive files cannot directly affect the operation of the workstation or ingest software; they can only change the way a localization is built.

Here is a summary of all of the directives that will be recognized in the LLL-mainConfig.txt file.

<b>Tag</b>	<b>Possible Values</b>	<b>Effect</b>
WFO	ID of any CWA	Specifies which county warning area is the basis for defining the geographic characteristics of the localization. Defaults to the localization ID.
CLONE	ID of any other viable localization.	Specifies that this localization shall be exactly like the one specified, except for whatever additional site-specific control files are added.
REALIZATION	RFC or NC	If RFC, then will build a localization suitable for a River Forecast Center. If NC, then a National Centers localization will be built. If no realization is defined, will build the default WFO realization.
SKEWT_TMIN	a Celsius temperature	Allows one to override the default temperature (-36.6) of the lower left corner of skew-T diagrams. The temperature of the lower right corner will change by the same amount, leaving the range of temperatures unchanged.
MAPDENS	a density value	Allows one to override the default minimum density setting (1) at which the inset map on skew-Ts, var vs. height displays, cross sections, etc. is visible. If global density is set lower than this value, the inset map is suppressed.
SATEW	EAST or WEST	Allows one to override the default satellite to ingest.
CCC	any AFOS CCC	Allows one to override the default AFOS CCC used by the text database.
CCC2	any AFOS CCC	In the case where the set of CCCs used to issue warnGen products is not uniform, this allows one to set a second CCC to possibly use. By default this is undefined...its proper utilization depends on having the correct entries in afosMasterPIL.txt file.
WMO	any WMO ID	Allows one to override the default WMO ID placed in the

		header of text products by the text editor. Defaults to KCCC.
XXX	one AFOS XXX	Establishes that AFOS XXX that is usually assigned to products generated at this site. Defaults to the localization ID.
XXX2	any AFOS XXX	In the case where the set of XXXs used to issue warnGen products is not uniform, this allows one to set a second XXX to possibly use. By default this is undefined...its proper utilization depends on having the correct entries in afosMasterPIL.txt file.
XXXL	list of AFOS XXXs	Establishes those AFOS XXXs associated with text products important to the local area. Implicitly includes that one established in the XXX directive. Can and should be a mix of 3 character XXXs and state IDs. Currently is primary means of determining which text products are used to make the Flash Flood Guidance graphic.
XXXR	list of AFOS XXXs	Establishes those AFOS XXXs associated with text products important to the regional area. Implicitly includes those in XXXL. Can and should be a mix of 3 character XXXs and state IDs. Currently is primary means of determining which text products are used to make the Local Warnings graphic.
NEW_FOG	TRUE or anything else	If TRUE, use Release 4.0/4.1 scaling for the `Fog' product. Otherwise use the Release 3.1 scaling. Defaults to 3.1 scaling.
HOME_SCALES	list of scale indices	If present, then this list of scales is set up to allow geographic radar selection through the `Home' menu.
ALL_HOME	TRUE or anything else	If TRUE, then all SBN radars will automatically be added to ingest and become available for geographic radar selection through the `Home' menu. This directive will have an effect only at an RFC or National Center.
RANGE_CUTOFF	a number from 200 to 500	Radial radar data will not be displayed further than this distance in kilometers from the radar. This helps some with the speed of display of radar data. Defaults to 460.
PUP_TABLES	TRUE or anything else	If TRUE, use PUP (OSF) color tables for radar products (defaults to FALSE).
RADAR_Z	color table index	Color table to use for 4 bit (16 color) radar reflectivity products. Default is determined by value of PUP_TABLES directive; see \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.
RADAR_3	color table	Color table to use for 3 bit (8 color) radar reflectivity

	index	products. Default is determined by value of PUP_TABLES directive; see \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.
RADAR_V	color table index	Color table to use for 4 bit (16 color) radar velocity products. Default is determined by value of PUP_TABLES directive; see \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.
RADAR_V3	color table index	Color table to use for 3 bit (8 color) radar velocity products. Default is determined by value of PUP_TABLES directive; see \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.
RADAR_P	color table index	Color table to use for radar precipitation accumulation products. Default is determined by value of PUP_TABLES directive; see \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.
RADAR_R	color table index	Color table to use for storm total precipitation products. Both FSL and OSF versions of this one are ramped by default to take advantage of the eight bit digital storm total precip. Default is determined by value of PUP_TABLES directive; see \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.
RADAR_W	color table index	Color table to use for radar spectrum width products. Default is determined by value of PUP_TABLES directive; see \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.
RADAR_S	color table index	Color table to use for radar radial shear products. Default is determined by value of PUP_TABLES directive; see

		<p>\$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.</p> <p>Color table to use for radar reflectivity products in general, meant to cover range of reflectivities from bottom of clear air mode to top of storm mode. See \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.</p> <p>Color table appropriate for entire dynamic range of 8 bit (256 color) reflectivity radar products. OSF and FSL versions of this table are maintained, though the default system no longer uses this table. See \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.</p>
RADAR_8	color table index	
RADAR_H	color table index	
RADAR_VH	color table index	<p>Color table to use for velocity radar products in general. Both OSF and FSL versions of this table cover entire possible dynamic range of 8 bit (256 level) velocities, but in the typical range of 4 bit velocities retain that color scheme. See \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.</p> <p>Color table to use for digital vertically integrated liquid. Both OSF and FSL versions of this table are inverse log, so that the color scheme approximates that of the old 4 bit VIL. See \$FXA_HOME/data/colorMaps.mark for usable default color table indices. It is also allowable to use the index for a user-defined color table, and to use a comma-delimited combination of color table index and percent brightness.</p>
RADAR_D	color table index	
WFOMAX	a distance in km	Allows one to override the default maximum size (600) of the WFO scale.
STATESIZE	a distance in km	Allows one to override the default size (900) of the State scale.
STATEWEST	a distance in km	Allows one to override the default westward bias (100) of the State scale.
STATENORTH	a distance in km	Allows one to override the default northward bias (0) of the State scale.

REGSIZE	a distance in km	Allows one to override the default size (2000) of the Regional scale. One should be cautious about making it larger, because larger regional satellite sectors will be stored, with possible disk utilization problems.
REGWEST	a distance in km	Allows one to override the default westward bias (200) of the Regional scale.
REGNORTH	a distance in km	Allows one to override the default northward bias (0) of the Regional scale.
CO_PD	a distance in km	Progressive disclosure parameter to use for county names. Defaults to 45.
ZN_PD	a distance in km	Progressive disclosure parameter to use for zone numbers. Defaults to 35.
TOPO_SCALES	scale indices	Space-delimited list of scales for which to actually generate topo. Defaults to `0 2 4'.
CITY_SCALE	a scale index	Scale index for which to generate city Map background. Defaults to 3, normally the Regional scale.
AREA_SCALE	a scale index	Scale that determines clip area for certain data sets, including some hi-res grids and some map backgrounds. Defaults to 3, normally the Regional scale.
NATCNTYSUP	a sup file name	Determines the area covered by the national counties/marine zones table, which is mainly used to support the display of watches using the WOU text product. Defaults to conusScale.sup if that is in the scaleInfo.txt file; otherwise defaults to the sup file for the scale with index 2.
MARINE_SCALE	a scale index	Scale that determines area over which to build a table for creating the marine site and marine zone location lists. A `t' means use the same area for which the warnGen tables are built. Defaults to 3, normally the Regional scale.
BUFR_SCALE	a scale index	Scale that determines area over which to process model soundings. For National Centers or FSL on a Linux system, the default will be to ingest all model BUFR soundings. Otherwise, this defaults to 5, normally the WFO scale.
GOES_BUFR_SCALE	a scale index	Scale that determines area over which to process GOES derived soundings. For National Centers or FSL on a Linux system, the default will be to ingest all soundings. Otherwise, this defaults to 3, normally the Regional scale.
POES_BUFR_SCALE	a scale index	Scale that determines area over which to process POES derived soundings. For National Centers or FSL on a Linux system, the default will be to ingest all soundings. Otherwise, this defaults to 3, normally the Regional scale.
BOTTOM	1000, 925, or	Sets the lower bound for some layer products in model

LLL-wwaConfig.txt includes globally defined defaults for some of its possible directives established in localization/nationalData/wwaDefaults.txt. Most WWA directives result in direct text substitution in the processed warnGen product template files; only the immediately following have a functional impact on the way the localization works:

Tag	Possible Values	Effect
TDIM	A number of grid points or a grid spacing in km.	Value for the grid_size global keyword in the warnGen GELT script files, which establishes the resolution of the warnGen geographic tables; this defaults to 600. GELT script files usually live in in nationalData/ and have _gsf.txt on the end. See <a href="#">section 4.1 of newGELTmaker.doc.html</a> for more information.
ORABOUT	TRUE or anything else	If TRUE, invokes a feature whereby warnGen will describe the location of weather events in relation to major cities, in addition to the nearest city in the warned area.
SBID	list of potential service backup localization IDs	A list of service backup localization IDs, each individually in double quotes, comma delimited, to which warnGen can be switched to run for on the fly. This is what is also called 'full service backup.' Note that just adding the ID here is not sufficient to enable full service backup; one must run that localization as well. Use the special -WWA task identifier if all you wish to do with the localization is support warnGen backup.
CFROMZ	TRUE or anything else	If TRUE, invokes a feature whereby the warnGen county table will be generated from forecast zone data.
AREA_SCALE	a scale index	Scale that determines area over which to build a table for creating the zone number and county name location lists. A 't' means use the same area for which the warnGen tables are built. If this same directive appears in mainConfig, will use that, otherwise defaults to 3, normally the Regional scale.
MARINE_SCALE	a scale index	Scale that determines area over which to build a table for creating the marine site and marine zone location lists. A 't' means use the same area for which the warnGen tables are built. If this same directive appears in mainConfig, will use that, otherwise defaults to 3, normally the Regional scale.

The unprocessed warnGen product definition templates live in localization/nationalData, and have .preWWA extensions. Localization uses these to create .wwaProd files, which are the final processed warnGen product template files, in the localization data set. The rest of these directives just result in a direct text substitution in the warnGen product definition files. The location of these substitutions is denoted by an '@@tag' string that looks just like the first word

of the directive in the LLL-wwaConfig.txt file. Most of these directives just default to an empty string if not present; any differences will be noted.

<b>Tag</b>	<b>Purpose</b>
OFFH	Name of the office to be used in the product header.
OFFT	Name of the office to be used in the text of a product.
AREAS	A term used to generically describe what a list of geographic entities is. Defaults to 'AREAS'.
COPE	How and whether to describe portions of areas for counties. Must be one or more of the following strings: ' portions', ' extreme', or ' central'. See <a href="#">TextTemplate.html</a> for more details.
ZOPE	How and whether to describe portions of areas for zones. Must be one or more of the following strings: ' portions', ' extreme', or ' central'. See <a href="#">TextTemplate.html</a> for more details.
CIPE	How and whether to describe portions of areas for cities being listed as locations within the warned area. Must be one or more of the following strings: ' portions', ' extreme', or ' central'. See <a href="#">TextTemplate.html</a> for more details.
PTPE	How and whether to describe portions of areas for cities used as a reference to describe the location of a storm. Must be one or more of the following strings: ' portions', ' extreme', or ' central'. See <a href="#">TextTemplate.html</a> for more details.
COFA	Establishes thresholds for automatically excluding small fragments of counties in the warnGen box. Must be one or more of the following strings: ' min_area=aaa', ' min_fraction=fff', or ' test_both', where aaa is an area in square km and fff is a fraction of a county. See <a href="#">TextTemplate.html</a> for more details.
ZOFA	Establishes thresholds for automatically excluding small fragments of zones in the warnGen box. Must be one or more of the following strings: ' min_area=aaa', ' min_fraction=fff', or ' test_both', where aaa is an area in square km and fff is a fraction of a zone. See <a href="#">TextTemplate.html</a> for more details.
CIFA	Establishes thresholds for automatically excluding small fragments of cities in the warnGen box. Must be one or more of the following strings: ' min_area=aaa', ' min_fraction=fff', or ' test_both', where aaa is an area in square km and fff is a fraction of a city. See <a href="#">TextTemplate.html</a> for more details.

---

**Author: Jim Ramer**  
**Last update: 15 Feb 08**

# Model Families

Model families are lists of gridded data displays that can be called up using a single button click from the main menu. Typically a model family consists of a list of eight overlays from a single model. The number eight is partly for historical reasons and partly because eight represents one graphic and one image in each panel of a 4-panel display. There are also some families that overlay the same field from several different models.

Model families are implemented as multi-loads. They are similar to, but not the same as, bundles. Anytime a user can load more than one overlay (excluding associated map backgrounds) with a single button click off of the main menu, this is a multi-load. Other examples are radar Z/V combos and satellite four panels. The main file that controls multi-loads is \$FXA\_HOME/data/multiLoadInfo.txt. This also has several #include statements that bring in radar, locally defined, non-FSL developed, and gridded data multi-loads. In order to make a multi-load displayable, one has to make entries for it in the depict keys, product buttons, and data menus as well.

In early versions of AWIPS, families were created by making manual entries in multiLoadInfo.txt, as well as in depictInfo.manual, productButtonInfo.txt, and dataMenus.txt (all in nationalData/). As the number of gridded data sources has grown, this has become an increasingly tedious exercise, and so it has been automated. It is now possible to make entries in the virtual field table (nationalData/virtualFieldTable.txt) and have them define a family for any new gridded data source that comes along. This document will just describe those elements of the virtual field table that have a direct bearing on model family generation; see [gridTables](#) for a complete description.

Model families are implemented in the virtual field table by a function called, appropriately, `MultiLoad'. Here are some examples of MultiLoad function entries; the first block is an example of part of the default family entry from 5.2.1 and previous, the second block is the whole default family entry from 5.2.2.:

```
ModFam| |N|Family| |OTHER| | | \
*MultiLoad,Layer|1.|GH,500MB|1.|AV,500MB \
|0.|msl-P,Surface|0.|dZ,1000MB-500MB \
|0.|GH,700MB|0.|RH,Layer \
|0.|PVV,700MB|0.|TP,Surface |0.|1.|2.|3.| \
*MultiLoad,Layer|1.|GH,500MB|1.|AV,500MB \
|0.|msl-P,Surface|0.|dZ,1000MB-500MB \
|0.|GH,700MB|0.|RH,@@BOTTOMMB-500MB \
|0.|PVV,700MB|0.|TP,Surface |0.|1.|2.|3.| \
```

```
ModFam| |N|Family| |OTHER| | | \
*MultiLoad,Layer|1.|GH,500MB|1.|AV,500MB|geoVort,500MB \
|0.|msl-P,Surface|msl-P2,Surface|0.|dZ,1000MB-500MB \
|0.1|GH,700MB|0.1|RH,Layer|RH,@@BOTTOMMB-500MB \
|0.1|PVV,700MB|wSp,250MB|P,Trop \
|0.1|TP3hr,Surface|TP6hr,Surface|TP,Surface \
|0.|1.|2.|3.
```

The most important thing to note is that in order to be treated as a family, and be posted to the main menu, the last word in the legend must be 'Family'. If this is not the case, one can still make this multi-load displayable in the volume browser by placing an entry for the variable ID ('ModFam' in this case) in \$FXA\_HOME/data/vb/browserFieldMenu.txt. Also note that after the function designation ('\*MultiLoad') there is a grid level name. With families, one generally wants to assign it to one and only one level. Since the result of a multi-load can be a mix of several different display types, using the 'OTHER' display type is most appropriate.

Each MultiLoad function entry consists of a list of overlays followed by an optional list of valid scale indices. Each overlay consists of two input variables; the variable to display preceded by a constant that describes how to display that variable. The display method constant is converted to the nearest integer before being interpreted as follows: A non-zero value in the ones place means this overlay should be toggled on by default. The tens digit is the display type to use: 0=contour, 1=icons, 2=image, 3=barbs, 4=streamlines, 5=arrows, 6=dualarrows, 7=other. A non-zero value in the hundreds digit means start a new pane. The thousands place is number of frames to load; 0 means the same as the number of forecast times and 99 means whatever the display is currently set for.

There are two important differences for 5.2.2. First, a non-integral entry in the constants for each overlay means that this component can be missing and still allow that particular MultiLoad function entry to be used. The other important difference is that one can place any number of field/level entries after each overlay constant. The first one that can be resolved for the source will be used. Note that the net result of this is a far more compact entry for the default model family.

Note that there are two MultiLoad function entries in the first block of these examples and that the last line ends in a continuation, meaning that there are more to come. For any gridded data source, an attempt will be made to create the multi-load with the first entry. To be successful, all of the variables must be available for display and the source must be available for display on one of the scales listed (scale considerations are ignored if no list is provided). If that fails it will try the next, and so on, until a usable function entry is found or it runs out of entries. If no entry is usable, then that multi-load will not be made for that source.

Entries in the virtual field table can only create model families that are for one gridded data source. There is another file that controls the creation of comparison families. The default version of that file is localization/nationalData/comparisonFields.txt. Each line in the file that is not blank or comments represents one field that a comparison family can be made out of. Each line is from two to five vertical bar delimited fields. At a minimum, a level ID and virtual field ID must be supplied. Optionally, one may supply a title in the third field, a comma delimited list of scales indices in the fourth, and a comma delimited list of time resolutions in the fifth. For each entry in comparisonFields.txt a multi-load can potentially be generated for each scale. If a list of scales is provided, then only those scales will have a multi-load generated. Each multi-load can potentially have an overlay from each gridded data source that has the variable in question. In order to be included, its default list of scales in the grid source table must include the scale in question. Also, if a list of time resolutions is provided, then the time resolution for the

source (15th field in gridSourceTable) must match one of the listed time resolutions. The overlays are sorted in decreasing order of length of forecast of the model.

In being posted into the main menu, families are categorized primarily by type, of which there are currently default, 4-panel, Surface, and comparison. Within each type, individual menu entries are made for each model, corresponding to all sources with the same legend title. For comparison families, individual entries are made for each field. The default type is automatically posted on the top level volume menu immediately below the product maker entry. The remaining types are posted on the top level menu if they have fewer than five entries, otherwise the entries for the type are all placed in a pull right.

**Author: Jim Ramer**  
**Last update: 11 Jan 02**

# File Dependencies

This document describes to the user what parts of the localization need to be rerun in response to changes in any given file in the source data for localizations.

All aspects of the localization process are described in more detail in the file [localization](#). Here we will summarize just enough about the workings of localization to support this discussion.

For the purpose of this discussion, `LLL' will always refer to the current localization ID, and `RRR' will always refer to the current realization ID, if applicable.

The phrase `source data for localizations' refers to four types of files: national data set files, site specific control files, realization files, and customization files. The following table shows pathnames to these various file types:

Type	Path(s)
national data	<code>\$FXA_HOME/data/*</code> <code>\$FXA_HOME/data/localization/nationalData/*</code>
site specific	<code>\$FXA_HOME/data/localization/LLL/LLL-*</code>
realization	<code>\$FXA_HOME/data/localization/realizations/RRR--*</code>
customization	<code>\$FXA_CUSTOM_FILES/*</code> <code>\$FXA_CUSTOM_FILES/LLL-*</code> <code>\$FXA_CUSTOM_FILES/\$FXA_CUSTOM_VERSION/*</code> <code>\$FXA_CUSTOM_FILES/\$FXA_CUSTOM_VERSION/LLL-*</code>

The user should note that `FXA_CUSTOM_FILES` currently points to `$FXA_DATA/customFiles` by default.

The discussion that follows covers all of the files in these categories that might be changed to effect the workings of a localization. Any file name mentioned is a possible replacement for a \* in the list above. Any file not in the national data set (site specific, realization, or customization) will be referred to generically as an `override file'.

An additional set of files important to this discussion is the localization data set files. These files are in `$FXA_HOME/data/localizationDataSets/LLL`. These are all of the site-unique files that are actually read by the workstation or ingest at run time to affect the way they behave. It is the function of the localization process to use the contents of the national data set and any applicable override files to correctly create the localization data set for the site.

It is important to note that only national data set files and localization data set files can change the way the AWIPS software functions. Override files can only affect the way a localization runs.

When one is doing a search for a specific file name in this document, remember that some files are listed with wild cards. For example, the file `raobMenus.txt` might be found under `raob\*.txt` or `\*Menus.txt`, so occasionally you might need to be creative with your search strings.

The rest of this document is broken up into four sections. The first section discusses localization config files. The second section discusses overriding the functionality of localization scripts. The third section discusses all other override files besides localization config files. The final section discusses the files in the national data set.

## 1) Localization config files

The files mainConfig.txt and wwaConfig.txt are a special case for this discussion (see [directives](#) for more information). Most override files affect one or two specific aspects of the localization. The file mainConfig.txt can affect any part of the localization. The file wwaConfig.txt is the same type of file as mainConfig.txt, but directives changed there can affect only the `wwa` task (rarely also the `station` task) of the localization. Thus, changes to wwaConfig.txt will result in having to run localizations only on workstation machines. Currently the only directives in wwaConfig.txt that can affect the `station` task are `AREA\_SCALE` and `MARINE\_SCALE`.

Here is a table of all of the currently recognized directives in mainConfig.txt and the localization tasks that one must run to implement that directive. Task names may be followed with some characters in parentheses. A `W` means that task need only be rerun on workstation machines, and a `d` means that one may choose to defer that task as not running it will not cause anything to fail, some things may just work slightly differently.

Directive	Task(s)
-----	-----
WFO	all tasks
CLONE	all tasks
REALIZATION	all tasks
SATEW	scales, clipSups, tables, maps(dW), station(dW), grids(dW)
SKEWT_TMIN	scales(W), dataSups(W),
CCC	text, wwa(W)
CCC2	wwa(W)
WMO	text
XXX	text, wwa(W)
XXX2	wwa(W)
XXXL	text
XXXR	text
HOME_SCALES	radar(W)
PUP_TABLES	radar(W)
RADAR_Z	radar(W)
RADAR_3	radar(W)
RADAR_V	radar(W)
RADAR_V3	radar(W)
RADAR_W	radar(W)
RADAR_S	radar(W)
RADAR_8	radar(W)
RADAR_H	radar(W)
RADAR_VH	radar(W)
RADAR_D	radar(W)

RADAR_P	radar(W)
RADAR_R	radar(W)
WFOMAX	scales(W), clipSups(W)
STATESIZE	scales(W), clipSups(W), topo(dW), station(dW)
STATEWEST	scales(W), clipSups(W), topo(dW), station(dW)
STATENORTH	scales(W), clipSups(W), topo(dW), station(dW)
REGSIZE	scales, clipSups, tables, maps(dW), station(dW), grids(dW)
REGWEST	scales, clipSups, tables, maps(dW), station(dW), grids(dW)
REGNORTH	scales, clipSups, tables, maps(dW), station(dW), grids(dW)
BUFR_SCALE	clipSups
GOES_BUFR_SCALE	clipSups
POES_BUFR_SCALE	clipSups
CO_PD	station(W)
ZN_PD	station(W)
TOPO_SCALES	topo(W)
CITY_SCALE	station(W)
AREA_SCALE	maps(W), station(W), wwa(W), grids(W)
MARINE_SCALE	station(W), wwa(W)
BOTTOM	grids

## 2) Script functionality override

The way the localization works, nearly all of the scripts that drive the localization live in \$FXA\_HOME/data/localization/scripts/. The script mainScript.csh is an executive that delegates the individual tasks involved in creating a localization data set to individual subordinate scripts. Each of the subordinate scripts can have its functionality either completely replaced or merely augmented by an override file. One can replace the entire functionality of a subordinate script only via a realization file, but augmenting a script's functionality is possible based on any kind of override file. For example, one of the subordinate scripts is called makeGridSourceTable.csh. The presence of a realization file called makeGridSourceTable.csh will cause the functionality of that script to be totally replaced. Any override file called makeGridSourceTable.patch will be sourced by the default makeGridSourceTable.csh in order to augment the functionality of that script. The manner in which .patch files are used has been made much more flexible in OB5; for more information on this, please see [scriptOverride](#).

Here is a list of all of the subordinate scripts that are subject to this type of override, along with that task that runs the script. One should note that the subordinate script makeScales.csh actually does have some functionality replaced rather than augmented by makeScales.patch.

Script	Task
-----	-----
makeGridSourceTable.csh	grids
makeDataSups.csh	dataSups
makeScales.csh	scales,clipSups
makeClipSups.csh	clipSups
assembleTables.csh	tables
makeTextKeys.csh	text
makeTopoFiles.csh	topo
updateGridFiles.csh	grids
updateRadarFiles.ksh	radar
makeMapFiles.csh	maps
makeWWAtables.csh	wwa

makeStationFiles.csh	station
makeDirectories.csh	dirs
createAuxFiles.csh	auxFiles
makePurgeTables.csh	purge

The following subordinate scripts can have their functionality replaced, but will not source a .patch file.

Script	Task
-----	-----
genRadarDataMenus.ksh	radar
genRadarDataKeys.ksh	radar
genRadarDepictKeys.ksh	radar
makeRadarSupps.csh	radar
genRadarProdButtonInfo.ksh	radar
genRadarMultiLoadKeys.ksh	radar
genRadarExtensionInfo.ksh	radar
doMosaicProcessing.ksh	radar
staUtil.csh	station
wwaUtil.csh	wwa
fixGridGeo.csh	fixGeo
fxatextTriggerConfig.sh	trigger

### 3) Other override files

This section describes which localization tasks need to be rerun when an override file other than mainConfig.txt or wwaConfig.txt is added or changed. This information is presented as a large table. For each override file, the national file column is that file in the national data set whose function this file is overriding or augmenting, if applicable. One should assume files in the national data sets are from localization/nationalData unless their name is preceded with a `data/', in which case they will be found in \$FXA\_HOME/data. A leading plus sign on a file means that it does not actually exist in the national data set but is generated during the localization by default. A leading minus sign means that the file does not actually exist in the national data set but there is something in the localization scripts or the national data set up to respond to that particular file. Where wildcards are used to indicate groups of files that are all handled alike, a list of all of the currently existing national files follows.

At times, neither the override file nor the national data set file whose functionality is being overridden will reflect the name of the file that is finally created or altered in the localization data set. When this occurs, the localization data set file name will be preceded by a `>'. This type of behavior will be noted only when the localization data set file created contains essentially the same information that was in the override file.

The tasks column is the list of localization tasks that need to be run to implement the change. The notations in parentheses are as before; a `W' means that task need be rerun only on workstation machines, and a `d' means that one may choose to defer that task. In addition, an `I' means that task need be rerun only on ingest machines, and an `A' means that task need be rerun only on the application servers. The task `others' will refer to any localization task normally run for a machine not already specifically mentioned. Whenever the `others' task is listed, it is

assumed that the national file column will contain only the name of the files most directly affected by the given override file; in reality many other files will potentially be affected.

The last three columns show the type of file override for this file. One should read [section 5.0 of localization](#) for more information on file override. `R` refers to realization files, `S` to site specific control files, and `C` to customization files. An empty column means that this file will not do anything for this category of localization source data. An `f` means it is subject to functional override, an `x` means it is subject to copy or replacement override, and an `a` means it is subject to append override. An `i` means that the override occurs through a `#include` in another file. This ends up working much like append override, with some slight differences that will be discussed. When include override is in effect, the `>` symbol is sometimes used to refer to the name of the file that is including the file in question. One should note that most localizations are not subject to a realization and as such the `R` column is not applicable. Also, be aware that sometimes the exact kind of override is somewhat ill defined or maybe some sort of hybrid. In that case the type of override that most closely applies is used.

With many override files that are plain ASCII, there is an opportunity to change the default type of override. If one adds a single line to the beginning of a file that says either `#replace` or `#append`, one can generally force the localization to bring that file into the localization with either replacement or append override, as the case may be. There are some exceptions. It is usually not possible to change a file that is subject to functional override (the `f` in the RCS column) to append override, and it not possible to change a file that is subject to include override (the `i` in the RCS column) to replace the contents of the file that is including it.

Another thing to keep in mind about file override is the way that most of the keyed files in AWIPS work. Keyed files are the files that have one entry per line, the lines being broken up into several vertical bar delimited fields, the first field being the key. These keys can be either numeric or text. Examples are depictInfo.manual and appInfo.txt. When the code that parses keyed files encounters entries with the same key, it will not cause an error. Rather, the last occurrence of an entry with that key will be the one that is used. If one is supplying an appended override file for a keyed file, it can still cause the replacement of the entries for individual keys that come before it in the final file.

override file	national file	tasks
RSC		
-----	-----	-----
--		
gridSourceTable.template xx.	gridSourceTable.template	grids,dirs(I)
	>gridSourceTable.txt	
tdlSourceTable.template xx.	tdlSourceTable.template	grids,dirs(I)
	>gridSourceTable.txt	
localGridSourceTable.txt .aa	gridSourceTable.template	grids,dirs(I)
	>gridSourceTable.txt	
*.wc	ecmf.wc	grids
fff		
	avn.wc	

	ukmet.wc	
	>gridSourceTable.txt	
activeGridSources.txt	activeGridSources.txt	grids,dirs(I)
xaa		
inactiveGridSources.txt	-inactiveGridSources.txt	grids,dirs(I)
xaa		
*.sup	+null.sup	dataSupS
xxx		
	+VAD.sup	
	+VWP.sup	
	+NUSM.sup	
	+conusC.sup	
	+nhSat.sup	
	+eastConus.sup	
	+westConus.sup	
	+libir.sup	
	+libvis.sup	
	+alaskaSat.sup	
	+akBigSat.sup	
	+hawaiiSat.sup	
	+hiBigSat.sup	
	+puertoRicoSat.sup	
	+prBigSat.sup	
	+conusA.sup	
	+llNorth.sup	
	+perspective.sup	
	+profTH.sup	
	+national.sup	
	+b03.sup	
	+b01.sup	
	+b02.sup	
grid*.sup	+grid201.sup	dataSupS,
xxx		
	+grid201.sup	grids
	+grid202.sup	
	+grid211.sup	
	+grid212.sup	
	+grid213.sup	
	+grid215.sup	
	+rucClip.sup	
	+gridmaps.sup	
	+maps40.sup	
	+grid203.sup	
	+grid207.sup	
	+grid204.sup	
	+grid205.sup	
	+grid214.sup	
	+msas.sup	
	+grid1.sup	
	+grid2.sup	
	+grid3.sup	
	+grid4.sup	
override file	national file	tasks
RSC		
-----	-----	-----
--		

cwa.bcd,cwa.asc .f.	usa_cwa.shx	scales,
	usa_cwa.shp.Z	wwa(W),others(d)
usa_cwa.* ff.	usa_cwa.dbf	scales,maps(W)
	usa_cwa.shx	
	usa_cwa.shp.Z	wwa(W),others(d)
	usa_cwa.dbf	
scaleInfo.txt xx.	scaleInfo.txt	scales,grids(W)
		grids(W)
baselines.static ..x	-baselines.static	scales(W)
points.static ..x	-points.static	scales(W)
depictInfo.manual x..	depictInfo.manual	tables
dataInfo.manual x..	dataInfo.manual	tables,dirs(I)
tdlDepictKeys.txt i..	tdlDepictKeys.txt	tables
	>depictInfo.manual	
tdlDataKeys.txt i..	tdlDataKeys.txt	tables,dirs(I)
	>dataInfo.manual	
localDepictKeys.txt .ia	>depictInfo.manual	tables
localDataKeys.txt .ia	>dataInfo.manual	tables,dirs(I)
redbook*s.txt ixx	redbookDataKeys.txt	tables,dirs(I)
	>dataInfo.manual	
	redbookDepictKeys.txt	
	>depictInfo.manual	
	redbookProductButtons.txt	
	>productButtonInfo.txt	
raob*s.txt .xx	raobDataKeys.txt	tables,dirs(I)
	raobDepictKeys.txt	
	raobProductButtons.txt	
msas*.txt .xx	msas_sysdef.txt	msas(A)
	>data/msas.cdl	
	>msasFieldConfig.txt	
	+msas.sup	
.ii	msasDepictKeys.txt	tables
	>depictInfo.manual	
.ii	msasProductButtons.txt	tables
	>productButtonInfo.txt	
.xx	msasFieldConfig.txt	tables
	>msasDepictKeys.txt	
*.config xaa	>msasProductButtons.txt	
	ipc.config	tables

	scales.config	
	tdl.config	
	ws.config	
*.abbrev	areas.abbrev	tables(W)
xaa		
	county_type.abbrev	
	state.abbrev	
satDataKeys.txt	-satDataKeys.txt	tables,dirs(I)
xxa		
satDepictKeys.txt	-satDepictKeys.txt	tables
xxa		
*SatDataInfo.template	westSatDataInfo.template	tables,dirs(I)
xx.		
	eastSatDataInfo.template	
	>satDataKeys.txt	
*SatDepictInfo.template	westSatDepictInfo.template	tables
xx.		
	eastSatDepictInfo.template	
	>satDepictKeys.txt	
imageStyle.txt	imageStyle.txt	tables(W)
x..		
tdlImageStyle.txt	tdlImageStyle.txt	tables(W)
x..		
localImageStyle.txt	imageStyle.txt	tables(W)
.aa		
productButtonInfo.txt	productButtonInfo.txt	tables(W)
x..		
tdlProductButtons.txt	tdlProductButtons.txt	tables(W)
x..		
satProductButtons.txt	satProductButtons.txt	tables(W)
xxa		
localProductButtons.txt	productButtonInfo.txt	tables(W)
.aa		
dataMenus.txt	dataMenus.txt	tables(W)
xxa		
*Menus.txt	backgroundMenus.txt	tables(W)
iia		
	commonLdadMenus.txt	
	redbookSurfaceMenus.txt	
	redbookUpperAirMenus.txt	
	raobMenus.txt	
	satDataMenus.txt	
	tdlAnalysisMenus.txt	
	tdlSurfaceMenus.txt	
	tdlToolMenus.txt	
	+raobLocalMenus.txt	
	-ldadMenus.txt	
	-otherAnalysisMenus.txt	
	-otherBackgroundMenus.txt	
	-otherHydroAppMenus.txt	
	-otherHydroProdMenus.txt	
	-otherSatMenus.txt	
	-otherSurfaceMenus.txt	
	-otherToolMenus.txt	
	-otherUaMenus.txt	
	-otherVolumeMenus.txt	
	-tdlBackgroundMenus.txt	

	-tdlRadarMenus.txt	
	-tdlUaMenus.txt	
	-tdlVolumeMenus.txt	
	>dataMenus.txt	
radarMenuHeader.txt	radarMenuHeader.txt	tables(W)
xxa		
radarMenuFooter.txt	radarMenuFooter.txt	tables(W)
xxa		
tdlMultiLoadInfo.txt	tdlMultiLoadInfo.txt	tables
i..		
	>data/multiLoadInfo.txt	
localMultiLoadInfo.txt	data/multiLoadInfo.txt	tables
.ia		
	>data/multiLoadInfo.txt	
localAppInfo.txt	data/appInfo.txt	tables(W)
.ia		
localExtensionInfo.txt	data/extensionInfo.txt	tables(W)
.ia		
*Design.txt	ldadMeso15Design.txt	tables
xxx		
	ldadMeso30Design.txt	
	ldadMesoHiWcDesign.txt	
	ldadPrcp15Design.txt	
	ldadPrcp1Design.txt	
	ldadPrcp30Design.txt	
	ldadPrcp3Design.txt	
	ldadQCallDesign.txt	
	ldadQCfailDesign.txt	
	maritimeMovingDesign.txt	
	maritimeStaticDesign.txt	
	maritimeStdDesign.txt	
	metar15Design.txt	
	metar24ChgDesign.txt	
	metarCvDesign.txt	
	metarHiWcDesign.txt	
	metarPrcp1Design.txt	
	metarPrcp24Design.txt	
	metarPrcp3Design.txt	
	metarPrcp6Design.txt	
	metarStdDesign.txt	
	msasAltFailDesign.txt	
	msasAltObsDesign.txt	
	msasDewdFailDesign.txt	
	msasDewdObsDesign.txt	
	msasDewpFailDesign.txt	
	msasDewpObsDesign.txt	
	msasPchgFailDesign.txt	
	msasPchgObsDesign.txt	
	msasPmsasFailDesign.txt	
	msasPmsasObsDesign.txt	
	msasPnwsFailDesign.txt	
	msasPnwsObsDesign.txt	
	msasThetaFailDesign.txt	
	msasThetaObsDesign.txt	
	msasWindFailDesign.txt	
	msasWindObsDesign.txt	
	profAglPlotDesign.txt	

```

profPerspPlotDesign.txt
profPresPlotDesign.txt
profSfcPlotDesign.txt
profTHdataDesign.txt
profVBdataDesign.txt
raobLowerDesign.txt
raobSkewtDesign.txt
raobUpperDesign.txt
*_*.txt      cloud_chars.txt      tables(W)
xxx

cloud_chars_nom.txt
cloud_select.txt
fractions_lookup.txt
hydro_acq_patterns.txt
maritime_cloud_chars.txt
prcp_formats.txt
prcp_goodness.txt
qc_check_bad.txt
qc_check_fmt.txt
qc_check_good.txt
rank_report_type.txt
raob_dd_char.txt
wx_symbol_trans.txt
colorMaps.mark data/colorMaps.mark      tables(W)
xx.

>data/colorMaps.nc
localColorMaps.mark data/colorMaps.mark      tables(W)
xxx

>localColorMaps.nc

override file      national file      tasks
RSC
-----
--
afosMasterPIL.txt      afosMasterPIL.txt      text
ff.

>afosMasterPIL.CCC
ispan_table.template      ispan_table.template      text
ff.

>ispan_table.dat
national_category_table.template      text
ff.

national_category_table.template
>national_category_table.dat

versions_lookup_table.template      text
ff.

versions_lookup_table.template
>versions_lookup_table.dat

versions_lookup_table.dat      text
.aa

versions_lookup_table.dat
textDataKeys.template      text,dirs(i)
ff.

>textDataKeys.txt
textDepictKeys.template      text
ff.

>textDepictKeys.txt

```

stateMatch.dat	stateMatch.dat	text
ff.		
textConfig.template	textConfig.template	text
ff.		
	>text.config	
text*Products.txt	textOfficeMenuProducts.txt	text
xxa		
	textAlarmAlertProducts.txt	
*.topo	data/ETA.topo	topo,grids
xxx		
	data/RUC.topo	
	data/grid202.topo	
	data/NGM.topo	
	data/MAPS.topo	
	data/grid201.topo	
	data/grid213.topo	
	data/MAPS40.topo	
	data/AVN211.topo	
*.cdl	data/aiv211.cdl	grids,
ff.		
	data/avn201.cdl	dirs(I)
	data/avn202.cdl	
	data/avn203.cdl	
	data/avn211.cdl	
	data/avn213.cdl	
	data/eta207.cdl	
	data/eta211.cdl	
	data/eta212.cdl	
	data/eta212.cdl	
	data/ecmfNH.cdl	
	data/laps.cdl	
	data/maps.cdl	
	data/maps40.cdl	
	data/mesoEta212.cdl	
	data/mesoEta215.cdl	
	data/mm5.cdl	
	data/mrf201.cdl	
	data/mrf202.cdl	
	data/mrf203.cdl	
	data/mrf204.cdl	
	data/mrf205.cdl	
	data/mrf213.cdl	
	data/msas.cdl	
	data/ngm202.cdl	
	data/ngm207.cdl	
	data/ngm211.cdl	
	data/ngm213.cdl	
	data/roc_rams.cdl	
	data/ruc211.cdl	
	data/sfm.cdl	
	data/turb212.cdl	
	>*.cdlTemplate	
override file	national file	tasks
RSC		
-----	-----	-----
--		

arrowStyle.rules xaa	arrowStyle.rules	grids(W)
tdlArrowStyle.rules a..	>arrowStyle.txt arrowStyle.rules	grids(W)
contourStyle.rules xaa	>arrowStyle.txt contourStyle.rules	grids(W)
tdlContourStyle.rules a..	>contourStyle.txt tdlContourStyle.rules	grids(W)
graphStyle.rules xaa	>contourStyle.txt graphStyle.rules	grids(W)
tdlGraphStyle.rules a..	>graphStyle.txt graphStyle.rules	grids(W)
gridImageStyle.rules xaa	>graphStyle.txt gridImageStyle.rules	grids(W)
tdlGridImageStyle.rules a..	>gridImageStyle.txt tdlGridImageStyle.rules	grids(W)
iconStyle.rules xaa	>gridImageStyle.txt iconStyle.rules	grids(W)
tdlIconStyle.rules a..	>iconStyle.txt iconStyle.rules	grids(W)
virtualFieldTable.txt xaa	>iconStyle.txt virtualFieldTable.txt	grids
tdlVirtualFieldTable.txt a..	tdlVirtualFieldTable.txt	grids
dataFieldTable.txt xaa	>virtualFieldTable.txt dataFieldTable.txt	grids
tdlDataFieldTable.txt a..	tdlDataFieldTable.txt	grids
gridPlaneTable.txt xaa	>dataFieldTable.txt gridPlaneTable.txt	grids
tdlGridPlaneTable.txt a..	tdlGridPlaneTable.txt	grids
dataLevelTypeTable.txt xaa	>gridPlaneTable.txt dataLevelTypeTable.txt	grids
tdlDataLevelTypeTable.txt a..	tdlDataLevelTypeTable.txt	grids
comparisonFields.txt fff	>dataLevelTypeTable.txt comparisonFields.txt	grids
browser*.txt xaa	data/vb/browserFieldMenu.txt	grids
	data/vb/browserPlanViewMenu.txt data/vb/browserSoundingFieldMenu.txt data/vb/browserSoundingMenu.txt	

```

data/vb/browserSpacePlanViewMenu.txt
data/vb/browserSpaceXSectionMenu.txt
data/vb/browserTimeHeightFieldMenu.txt
data/vb/browserTimeHeightMenu.txt
data/vb/browserXsectFieldMenu.txt
>vb/browser*.txt

override file          national file          tasks
RSC
-----
--
radarsInUse.txt       +radarsInUse.txt     radar
xxx
radarsOnMenu.txt     +radarsOnMenu.txt    radar
xxx
pupId.txt             +pupId.txt           radar(I)
xxx
dialPorts.txt        data/dialPorts.txt   radar(I)
xxx
portInfo.txt         +portInfo.txt        radar(I)
xxx
dialRadars.txt       +dialRadars.txt      radar(I)
xxx
mosaicScales.txt     +mosaicScales.txt    radar
xxx
radarDataKeys.template radarDataKeys.template radar,dirs(I)
faa
radar*.template      >radarDataKeys.txt
faa                  radarDepictKeys.template radar
radarMultiLoadInfo.template
>radar*.txt
radar*.template      -radarExtensionInfo.template radar(W)
faa
radarProductButtonInfo.template
>radar*.txt
radar*.template      radarDataMenus.template radar(W)
ffa
>radarDataMenus.auto
>radarDialMenus.auto
>homeDataMenus.auto
radar*StyleInfo.template radarImageStyleInfo.template radar(W)
aaa
>radarImageStyleInfo.txt
mosaic*.template     mosaicDepictKeys.template radar
faa
>mosaic*.txt
mosaic*.template     mosaicProductButtons.template radar(W)
faa
>mosaic*.txt
mosaic*.template     mosaicDataMenus.template radar(W)
faa
>mosaicDataMenus.auto
mosaicInfo.txt       +mosaicInfo.txt      radar
fff
radarTextProds.template radarTextProds.template radar
ff.

```

```

>afosMasterPIL.CCC
tdlRadarDataKeys.template radar,dirs(I)
ff.

tdlRadar*.template
ff.
tdlRadarDataKeys.template
>radarDataKeys.txt
tdlRadarDepictKeys.template radar

tdlRadar*.template
ff.
tdlRadarMultiLoadInfo.template
>radar*.txt
tdlRadarExtensionInfo.template radar(W)

tdlRadarDataMenus.template
tdlRadarProductButtonInfo.template
-tdlRadarImageStyleInfo.template
>radar*.txt
radarInfoMaster.txt radar
ff.
+radarInfoMaster.txt radar
radarInfoMaster.patch
aaa
c11-zone.*
ff.
c11-zone.shp.Z maps(W),
c11-zone.dbf wwa(W)
c11-zone.shx
uscounty.*
ff.
uscounty.shp.Z maps(W),
uscounty.dbf wwa(W)
uscounty.shx
basins.*
ff.
basins.shp.Z maps(W)
basins.dbf
basins.shx
usa_lake.*
ff.
usa_lake.shp.Z maps(W)
usa_lake.dbf
usa_lake.shx
usa_rfc.*
ff.
usa_rfc.shp.Z maps(W)
usa_rfc.dbf
usa_rfc.shx
us_inter.*
ff.
us_inter.shp.Z maps(W)
us_inter.dbf
us_inter.shx
*.bcd
xxx
data/artcchi.bcd
data/conandsta.bcd
data/countyPlus.bcd
data/latlon10.bcd
data/twebRoutes.bcd
+data/usa_cwa.bcd
+data/c11-zone.bcd
+reg_zones.bcd
+data/uscounty.bcd
+reg_county.bcd
+data/thin_county.bcd
+data/basins.bcd
+reg_lake.bcd

```

*.bcx	+data/usa_rfc.bcd	
xxx	us_inter.bcx	maps(W)
override file	national file	tasks
RSC		
-----	-----	-----
--		
usa_wsfo.*	usa_wsfo.shp.Z	wwa(W)
ff.		
	usa_wsfo.dbf	
	usa_wsfo.shx	
wsfo.bcd,wsfo.asc	usa_wsfo.dbf	wwa(W)
.f.		
	usa_wsfo.dbx	
	usa_wsfo.shp.Z	
usa_cwa_total.*	usa_cwa_total.shp.Z	wwa(W)
ff.		
	usa_cwa_total.dbf	
	usa_cwa_total.shx	
cwaTotal.bcd,	usa_cwa_total.dbf	wwa(W)
.f.		
cwaTotal.asc	usa_cwa_total.dbx	
	usa_cwa_total.shp.Z	
urban_bounds.*	urban_bounds.shp.Z	wwa(W)
ff.		
	urban_bounds.dbf	
	urban_bounds.shx	
wwa_counties_spec.txt	uscounty.{shp.Z,dbf,shx}	wwa(W)
fff		
	>wwa_counties.{gelt,id,entity,table,NS,EW}	
wwa_warn_city_spec.txt	urban_bounds.{shp.Z,dbf,shx}	wwa(W)
fff		
	>wwa_warn_city.{gelt,id,entity,table,NS,EW}	
marine_zones.*	marine_zones.dbf	wwa(W)
ff.		
	marine_zones.dbx	
	marine_zones.shp.Z	
marineArea.bcd	marine_zones.shp	wwa(W)
.f.		
marine_total.*	marine_total.shp.Z	wwa(W)
ff.		
	marine_total.dbf	
	marine_total.shx	
timezones.*	timezones.dbf	wwa(W)
ff.		
	timezones.dbx	
	timezones.shp	
CitiesInfo.txt	data/CitiesInfo.txt	wwa(W),
ff.		
		station(W)
LocalCitiesInfo.txt	data/CitiesInfo.txt	wwa(W),
.aa		
		station(W)
wwa_warn_city.bcd		wwa(W)
.f.		

MarineInfo.txt xx.	MarineInfo.txt	wwa(W),  station(W)
wwa*.preTemplate f..	wwa_zones_ugc.preTemplate  wwa_zones_ugc.preTemplate wwa_wsfo_zone_ugc.preTemplate wwa_county_ugc.preTemplate wwa_wsfo_cnty_ugc.preTemplate wwa_marine_ugc.preTemplate >wwa*.template	wwa(W)
wwa*_block.preTemplate f..	wwa_county_block.preTemplate  wwa_zones_block.preTemplate wwa_marine_block.preTemplate >wwa*_list.template	wwa(W)
sls_county_all.preTemplate f..	>wwa_county_list.template	wwa(W)
sls_county_block.preTemplate f..	>wwa_county_list.template	wwa(W)
wwa*.preWWA fff	wwa_blizzard_wrn.preWWA  wwa_blodust_adv.preWWA wwa_blosnow_adv.preWWA wwa_dam_break.preWWA wwa_ffw.preWWA wwa_flflood_sta.preWWA wwa_flflood_wat.preWWA wwa_flood_wat.preWWA wwa_flood_wrn.preWWA wwa_fog_adv.preWWA wwa_freez_wrn.preWWA wwa_frost_adv.preWWA wwa_frzdrzl_adv.preWWA wwa_frzrain_adv.preWWA wwa_heat_warn.preWWA wwa_hiwind_wat.preWWA wwa_hiwind_wrn.preWWA wwa_hvysnow_wrn.preWWA wwa_shortterm_fcst.preWWA wwa_snoblsno_adv.preWWA wwa_snow_adv.preWWA wwa_specmarine.preWWA wwa_svr.preWWA wwa_svrhiwind_wrn.preWWA wwa_svrwx_sta.preWWA wwa_sws.preWWA wwa_sws_county.preWWA wwa_tor.preWWA wwa_urbssflood_adv.preWWA wwa_wintstrm_wat.preWWA wwa_wintstrm_wrn.preWWA wwa_wndchil_adv.preWWA >wwa*.wwaProd	wwa(W)

wwaConfig.template ff.	wwaConfig.template	wwa(W)
wwa.config .xx	>wwa.config wwaConfig.template	wwa(W)
*_gsf.txt fff	>wwa.config CWA_USA_gsf.txt	wwa(W)
	backupCWA_gsf.txt backupMarine_gsf.txt cwaTotalBcd_gsf.txt cwaTotal_gsf.txt marineTotal_gsf.txt marine_zones_gsf.txt reg_counties_gsf.txt reg_marine_gsf.txt reg_zones_gsf.txt wsfoTableBcd_gsf.txt wsfoTable_gsf.txt wwa_countiesZ_gsf.txt wwa_counties_gsf.txt wwa_marine_sites_gsf.txt wwa_marine_sites_id_gsf.txt wwa_timezone_gsf.txt wwa_warn_city_gsf.txt wwa_warn_city_id_gsf.txt wwa_zones_gsf.txt >*. {gelt, id, entity, table, NS, EW}	
override file RSC	national file	tasks
----- --	-----	-----
raob.goodness ffa	raob.goodness	dataSups(W),
*.goodness ffa	BUOY.goodness	station(W) station(W)
	profiler.goodness twebRoutes.goodness twebStations.goodness data/MTR.goodness +88D.goodness +cities.goodness +marine_sites.goodness +warn_cities.goodness +ldad15prcp.goodness +ldad15.goodness >*. {spi, lpi}	
*.spi ffa	+BUOY.spi	station(W)
	+profiler.spi +raob.spi +MTR.spi +ldad15prcp.spi +ldad15.spi	

*.lpi	latlon10.lpi	station(W)
ffa		
	+88D.lpi	
	+cities.lpi	
	+twebRoutes.lpi	
	+twebStations.lpi	
	+warn_cities.lpi	
	+counties.lpi	
	+zone_nums.lpi	
	+marine_nums.lpi	
	+marine_sites.lpi	
	+cwa_usa.lpi	
*.primary	profiler.primary	station(W)
aaa		
	data/MTR.primary	
	raob.primary	
	-BUOY.primary	
	-twebRoutes.primary	
	-twebStations.primary	
	-88D.primary	
	-cities.primary	
	-warn_cities.primary	
	-ldad15prcp.primary	
	-ldad15.primary	
*.cdl	LDADhydro.cdl	dirs(I)
fff		
	LDADmanual.cdl	
	LDADmesonet.cdl	
	ldad_qcobs.cdl	
	maritime.cdl	
	metar.cdl	
	msas_qcobs.cdl	
	profiler.cdl	
	raob.cdl	
*.cdl	spotters.cdl	station(W)
fff		
	>*.nc	
*.dat	spotters.dat	station(W)
fff		
	>*.nc	
*.dat	+*Seed.dat	station(W)
f..		
	>*.nc	
progDiscStnTables.txt	+progDiscStnTables.txt	station(W)
faa		
*GoodnessDesign.txt	maritimeGoodnessDesign.txt	station(W)
fff		
*MenuItems.txt	removeMenuItems.txt	tables,radar(W)
xaa		
	-preserveMenuItems.txt	
acq_patterns.template	acq_patterns.template	auxFiles(I)
.x.		
	>acq_patterns.txt	
KXXX.*	data/KXXX.clear-air	auxFiles(I)
xx.		
	data/KXXX.storm	

ldadSiteConfig.txt	ldadTrigger.template	trigger(I)
.x.		
hydroSiteConfig.txt	hydroTrigger.template	trigger(I)
.x.		
override file	national file	tasks
RSC		
-----	-----	-----
--		
purgeInfo.txt	purgeInfo.txt	purge(I)
xa.		
radarPurgeInfo.txt,	radarPurgeInfo.txt	purge(I)
ia.		
gridPurgeInfo.txt,	gridPurgeInfo.txt	
satPurgeInfo.txt	satPurgeInfo.txt	
	>purgeInfo.txt	
localPurgeInfo.txt	localPurgeInfo.txt	purge(I)
.ia		
	>purgeInfo.txt	
radarPurgeInfo.template,	radarPurgeInfo.txt	purge(I)
faa		
dialPurgeInfo.template		

#### 4) National data set files

This section contains a large table that says which localization tasks need to be rerun as a result of changing any file in the national data set. Normally, one would assume that a change to the national data set would occur only as the result of a software upgrade or per the instructions of the NCF or OOS. Much of the information in this table will be in some ways redundant to what was presented in the previous section.

As in the previous section, files in the national data sets are assumed to be from localization/nationalData unless their name is preceded with a `data/', in which case they will be found in \$FXA\_HOME/data. Also as in the previous section, the notations in parentheses contain `W', which means that task need be rerun only on workstation machines, `I', which means that task need be rerun only on ingest machines, `A', which means that task need be rerun only on the application servers, and `d', which means that one may choose to defer that task. An x in the 'x' column means that one need run a localization only if there is already a file by the same name in the localization data set. There are even some files in the national data set that can be replaced and do not require one to run a localization; they always take effect with just a process restart.

file(s)	x task(s)
-----	-----
gridSourceTable.template	grids,dirs(I)
tdlSourceTable.template	grids,dirs(I)
activeGridSources.txt	grids,dirs(I)
tdlActiveGridSources.txt	grids,dirs(I)
usa_cwa.*	scales, clipSupps, tables, maps(dW), station(dW), grids(dW)
scaleInfo.txt	x scales

depictInfo.manual	x tables
dataInfo.manual	x tables,dirs(I)
tdlDepictKeys.txt	x tables
tdlDataKeys.txt	x tables,dirs(I)
tdlExtensionInfo.txt	x
*.config	x tables
*.abbrev	x tables
*SatDataInfo.template	tables,dirs(I)
*SatDepictInfo.template	tables
imageStyle.txt	x tables(W)
tdlImageStyle.txt	x tables(W)
productButtonInfo.txt	x tables(W)
tdlProductButtons.txt	x tables(W)
satProductButtons.txt	x tables(W)
*Menus.txt	x tables(W)
*MenuFooter.txt	x tables(W)
*MenuHeader.txt	x tables(W)
data/multiLoadInfo.txt	x tables
tdlMultiLoadInfo.txt	x tables(W)
data/colorMaps.mark	tables(W)
afosMasterPIL.txt	text
*_table.template	text
textD*Keys.template	text
cccLatLon*.txt	text
stateMatch.dat	text
textConfig.template	text
text*Products.txt	x text
*Topo.dat.gz	topo(W),topo(d),grids(d)
data/*.topo	topo,grids
data/*.cdl	grids,dirs(I)
*.rules	grids(W)
*FieldTable.txt	grids
*PlaneTable.txt	grids
*LevelTypeTable.txt	grids
data/vb/browser*Menu.txt	x grids(W)
fsl-w88d.*	radar
data/dialPorts.txt	x
radarDataKeys.template	radar,dirs(I)
radarDepictKeys.template	radar
radarMultiLoadInfo.template	radar
radarTextProds.template	radar
radarDataMenus.template	radar(W)
radarProductButtonInfo.template	radar(W)
tdlRadarDataKeys.template	radar,dirs(I)
tdlRadarDepictKeys.template	radar
tdlRadarMultiLoadInfo.template	radar
tdlRadarExtensionInfo.template	radar(W)
tdlRadarDataMenus.template	radar(W)
tdlRadarProductButtonInfo.template	radar(W)
mosaicDepictKeys.template	radar
mosaicDataMenus.template	radar(W)
mosaicProductButtons.template	radar(W)
c11-zone.*	maps(W),wwa(W)
uscounty.*	maps(W),wwa(W)
basins.*	maps(W)
usa_lake.*	maps(W)
usa_rfc.*	maps(W)

us_inter.*	maps(W)
*.bcd	x maps(W)
usa_wsfo.*	wwa(W)
usa_cwa_total.*	wwa(W)
marine_zones.*	wwa(W)
timezones.*	wwa(W)
urban_bounds.*	wwa(W)
marine_total.*	wwa(W)
data/CitiesInfo.txt	wwa(W),station(W)
MarineInfo.txt	wwa(W),station(W)
wwaDefaults.txt	wwa(W)
wwa*.preTemplate	wwa(W)
sls*.preTemplate	wwa(W)
wwa_*.preWWA	wwa(W)
*_gsf.txt	wwa(W)
wwaConfig.template	wwa(W)
*.goodness	station(W)
*.primary	station(W)
*.lpi	x station(W)
data/metarStationInfo.txt	station(W)
data/KXXX.*	auxFiles(I)
*Trigger.template	trigger(I)
data/{a,selsA}nchors.txt	x
redbook*.txt	x tables
raobD*Keys.txt	x
raobProductButtons.txt	x
*.{cdl,dat}	station(W)
*urgeInfo*	purge(I)

---

**Author: Jim Ramer**  
**Last update: 17 Sep 04**

# Grid Tables

This file contains documentation for five files that are used to manage gridded data and the volume browser: [gridSourceTable.txt](#), [dataLevelTypeTable.txt](#), [gridPlaneTable.txt](#), [dataFieldTable.txt](#), and [virtualFieldTable.txt](#). These files reside in the directory `localization/nationalData`, which can be found in `$FXA_HOME/src` in an environment with a source tree or in `$FXA_HOME/data` otherwise. Once one has completed changes in these tables to one's satisfaction, they are implemented by running the ``-grids'` localization task.

The software that builds the data key and depictable key tables for gridded data interrogates the gridded data files to determine which combinations of sources, planes, and fields actually have data available in netCDF files. The relationships established in these five tables then determine which additional combinations can be made available by making [various calculations](#), and puts into the data key table information on exactly how these calculations should be performed. Also part of the localization's management of gridded data sets is the auto-generation of model families. For more information on this, see the description of the [MultiLoad](#) function in this document, or the document [families](#).

There are seven additional tables – `arrowStyle.rules`, `gridImageStyle.rules`, `contourStyle.rules`, `streamlineStyle.rules`, `graphStyle.rules`, `barbStyle.rules`, and `iconStyle.rules` – which control how to display gridded fields once they have been retrieved and/or calculated. See [styleRules](#).

As one reads this documentation, it is useful to refer to the files which are being described - it will make much more sense that way.

## The Source Table

The file `gridSourceTable.txt` is where gridded data sources are made known to the system. The copy of `gridSourceTable.txt` that is eventually used is generated by the localization scripts by processing `gridSourceTable.template`, `tdlSourceTable.template`, and one or more `localGridSourceTable.txt` files with `sed` commands.

An examination of `gridSourceTable.template` will reveal that there are 51 grid sources listed, followed by a line reading simply 100, then numerous additional sources. The reason for this is that grid sources are identified numerically in parts of the system, and ordering is therefore important. The '100' is an index mark, specifying that subsequent sources' index numbers start at 100.

The file `tdlSourceTable.template` is where non-core developers put their gridded data sources. The '51' on the first line causes the source indexing to start at 51.

The localization will also try to add locally-supplied file(s) `[LLL-]localGridSourceTable.txt` to the final grid source table. These can reside either in `/awips/fxa/data/localization/LLL` or in `$FXA_CUSTOM_FILES`. If only one of these exists, it should fix its indexing at 60, allowing room for nine sources in `tdlSourceTable` (only two are currently used). If both are used, the

CUSTOM\_FILES version must start its indexing high enough to avoid conflict with the other. In any case, local indexing must be cognizant of gridSourceTable.template's use of 100 - though it's unlikely that more than 40 sources will be needed locally.

A field user should not need to modify gridSourceTable.template or tdlSourceTable.template. However, in no case should the order of sources be changed in these files, nor should sources be added except at the end of the file. This same admonition holds for local grid sources, as well, because the index numbers are used in bundles, and any reordering may break procedures that include those sources.

Each line in the file contains the information for one source, and has 15 primary text fields separated by vertical bars. Here is a description of what each text field means. (There is also similar documentation in the gridSourceTable.template file.)

1. Alias. This allows the user to specify that, to the outside world, displays resulting from this gridded data source look like they are from some other gridded data source. Entries here must be the unique name (text field 10) of some other source. If an aliased source is inactive, it can still be used to direct the ingest of grids to the active source to which it is aliased.
2. Any non-blank character in this text field other than a number will cause keys not to be built for this data source. A number will designate which version of the GridAccessor is being used. The *sed* commands run on gridSourceTable.template to create gridSourceTable.txt will often operate on this field.
3. Directory path to netCDF files containing the gridded data for this source. This is the path minus the leading \$FXA\_DATA/. Multiple sources that have the same directory path will automatically be aliased to the first active source with that directory path.
4. CDL name. The CDL file which describes how to build the netCDF file has this name, plus a .cdl extension. A source cannot be activated if this field is blank. For point data sets presented by the volume browser, this is the name of the file that contains the master station list for geographic selection.
5. Geographic information file that describes the area covered by this grid. Must be one of the files matching the specification \$FXA\_DATA/\*sup. If a second comma-delimited geographic information file is included, the first describes the area covered by the data that are stored, and the second describes the area covered by the data that are received. No remapping is done to accomplish this; it is all by clipping or partial filling. The *sed* commands run on gridSourceTable.template to create gridSourceTable.txt will often operate on this field. For point data sets presented by the volume browser, this is the name of the design file used to gather the data.
6. X dimension of the grids that are stored. If negative, it is assumed that the dimensions are of the data that are received; this is applicable only if there is a second depictor file listed in the previous field. If the dimensions are of the data that are stored and they do not match those of the CDL, they will be adjusted.
7. Y dimension of the grids that are stored.
8. Title. This is how this source is referred to in the volume browser and for legends. This can be adapted based on entries in the activeGridSources.txt file. A title with an underscore will cause the source to be treated as a test source. This means it will appear

at the end of the source selection menus in the volume browser and any families from the source will always be in a pull right off of the main Volume menu.

9. List of scale indices, separated by commas, for which this source is valid. This is used in conjunction with a similar text field for cross section planes in the file gridPlaneTable.txt so that, for example, depict keys for WFO scale cross section planes are not generated for the hemispheric GFS grids. This also controls which sources appear in the volume browser menus for which scales. This can be adapted based on entries in the activeGridSources.txt file.
10. Unique name of this source. Aliases in the first field and names of sources to activate in activeGridSources.txt must refer to this entry.
11. Topography file. Must be one of the files matching the specification \$FXA\_DATA/\*topo. Topography grids are just flat ASCII files with a list of numbers, one per line. If an entry exists here and a corresponding \*.topo file does not exist, the localization will generate one.
12. GRIB grid identifier. As grids arrive, they are associated with an originating center according to an item in the GRIB header. The file \$FXA\_HOME/data/gribTableInfo.txt is a table of all currently known centers. The first file (column 2) for a center lists known grid IDs for the center. This entry must be one of these ASCII grid IDs for the appropriate center. To store a grid in the source, it must be for this grid and one of the process IDs listed in the next field.
13. GRIB process identifier(s). The second file (column 3) in gribTableInfo.txt for the originating center lists known model (process) IDs for the center. This entry must be one or more of these ASCII process IDs for the appropriate center, comma delimited. To store a grid in the source, it must be for one of these listed process IDs and the grid identified in the previous source.
14. Optional GRIB parameter identifier. The third file (column 4) in gribTableInfo.txt for the originating center lists known parameter IDs for the center. If non-blank, this entry must be one or more of these ASCII parameter IDs for the appropriate center, comma delimited, and only those parameters can be stored for the source. If blank, assumes any variable can be stored.
15. This is the intrinsic frequency of this data source. If less than 300, it is assumed to be in hours, otherwise in seconds. It should ideally be the minimum separation of valid times, not initial times.

In order to have a netCDF file available to display data for a new gridded data source, one needs to create a new .cdl file. To make a new gridded data source displayable, one needs to add it to activeGridSources.txt. In activeGridSources.txt, a list of comma-delimited integers immediately after a source name will be interpreted as a scale list, and will replace the scale list in the grid source table. A string with a leading `>' immediately after a source name will be interpreted as a new title for the source. Both adaptations can be present for a given source.

There are some cases where the grids we want to store for a data source actually arrive on the SBN in several smaller pieces. This is handled by having just one active source that is meant to present the data to the volume browser, and several additional inactive source used by the ingest to stitch in each piece. These additional inactive sources are generated automatically by the localization.

When an asterisk followed by a file name occurs in a source entry, that file is read to automatically create a source for each line in the file. The first line represents the source that remains in the table where it was originally entered. The remaining lines represent sources that are appended to the end of the table, and are left inactive. As each source is composed from the entry with the asterisks, the file name is replaced by the first space-delimited field in the line from the wild card file. Similarly, \*2 is replaced by the second field, \*3 the third, up to six fields. A field with just a period is interpreted as an empty string.

## **The Level Type Table**

The file dataLevelTypeTable.txt is where level types are made known to the system. Each line contains the information about one level type. Each line has three text fields separated by vertical bars. Here is a description of these fields:

1. The level type name, which is a unique identifier for this level type that is referred to in other tables.
2. Now obsolete.
3. The mode identifier for this level type. 0 means that the level type is non-parametric, like tropopause. A positive number means a parametric level type that increases upward, like height, and a negative number means a parametric level type that decreases upward, like pressure. If blank, then this level type is used to refer to non-plan view planes, like cross sections.

The user should note that the software internally generates five additional level types, which have the names XC\_LAT, XC\_LON, XC\_SPEC, PLACE\_HOLDER, and NULL\_TYPE.

## **The Plane Table**

The file gridPlaneTable.txt is where planes (levels) are made known to the system. Each line in the file contains the information about one plane. Each line has from 3 to 6 text fields separated by vertical bars. When new planes are added, they should always be added to the end of the file, or the plane indexing will be changed which can break things.

Field users should place just their new planes in a version of gridPlaneTable.txt that sits in their site-specific directory (localization/LLL/LLL-gridPlaneTable.txt) or their customization directory (\$FXA\_CUSTOM\_FILES/gridPlaneTable.txt). In order to fix the indexing, such a file should begin with a line containing just a number; 600 is a good choice.

Any vertical coordinate type used in the file must appear in the level type table described in the previous section. We refer to a vertical coordinate type and value as a level definition. Those vertical coordinate types which are non-parametric can constitute a level definition without a value.

There are two major categories of planes – plan-view planes and non plan-view planes. The two are different enough that they will be treated separately. Here is a description of each item in a plan-view plane entry:

1. Name of the plane. If blank, a name will be provided by default. For standard planes, the default name will be a concatenation of the text provided for the level value and the level type. For composite or binary planes, the default name will be a concatenation of the text provided for each level value and level type, the two separated by a hyphen. There is a special plane name called `spatial`. Any plane with this name is assumed to represent a situation where loops are constructed by stepping through a series of planes instead of a series of times.
2. Type of plane. For plan view planes, this should be either `standard`, `composite`, or `binary`. Standard planes are just that: the normal concept of a level identified by a single vertical coordinate type and value. Composite and binary planes are identified by two level definitions, a lower and upper one, like 1000mb-500mb. For composite planes, it is assumed that data for the plane are not found in the netCDF files, but are calculated from other variables on the upper and lower levels that make up the plane. Binary planes are identified by two level definitions, but data for them can actually be found in the netCDF files.
3. For standard planes, the level definition; for composite or binary planes, the first level definition. The level definition is a value followed by a level type name, comma delimited. For a non-parametric level, only the level type name is entered.
4. For composite or binary planes, this is the second level definition. For standard planes, an optional plane visibility parameter. The visibility parameter is either a dash that says that data from this plane may be used for calculations, but will not be selectable from the volume browser, or the name of some other plane, which means all data from this plane will automatically be treated as if they were from some other plane.
5. Optional plane visibility parameter for composite or binary planes. For standard planes, an optional grouping index, which defaults to zero. When planes are sorted, planes with the same plane type and level type are sorted together, primarily by grouping index and secondarily by level value.
6. Optional grouping index for composite or binary planes.

For non plan-view planes, a level definition does not have the same meaning as it does for plan-view planes. For non plan-view planes, planes with the same level definition are logically grouped together, differing only by their geographic information file. This is basically an efficiency hack, and it also allows one to set style info for all planes so grouped by just setting the style info for the first plane of the group. Here is a description of each item in a non plan-view plane entry:

1. Name of the plane. For latitude and longitude cross sections, this will be overridden and automatically generated. There is a special plane name called `spatial`. Any plane with this name is assumed to represent a situation where loops are constructed by stepping through a series of different baselines instead of a series of times.
2. Type of plane. For non plan-view planes this should be either `xsect`, `tsect`, `vrtgph`, or `diagram`. These refer to spatial cross sections, time-height cross sections, variable vs height graphs, and thermodynamic diagrams (gridded soundings), respectively. All planes of type `xsect` will have their volume browser menu entries generated automatically.

3. Level definition. As mentioned, this is used only to logically group planes together that have identical characteristics except for their geographic information file. For cross sections, this information is automatically replaced internally in the table.
4. Vertical coordinate type for the level definition.
5. Depictor file name. This controls the location to which data are interpolated for this particular non plan-view plane. If the plane type is `xsect' and the depictor name contains the sub-string `Lat' or `Lon', then this will become a latitude or longitude plane and the depictor file will be generated automatically. For other spatial cross section planes, the depictor name should refer to a file from which one can construct an XsectDepictor; this XsectDepictor describes the baseline of the cross section. For a time-height cross section, this should refer to an XsectDepictor file with only one lat/lon point. For thermodynamic diagram (sounding) planes, the depictor name should refer to a file from which one can construct a ThermoDepictor; this ThermoDepictor carries the latitude and longitude of the sounding, and also describes the thermodynamic background upon which the gridded sounding is drawn.
6. List of scale indices, separated by commas, for which this plane is valid. This is used in conjunction with a similar text field in the file gridSourceTable.txt so that, for example, depict keys for WFO scale cross section planes are not generated for the hemispheric GFS grids. For latitude or longitude cross section planes, there should be only one scale entered. The locations of these planes will then be automatically selected so that for however many planes there are on that scale, they will be evenly spaced to roughly cover the area of that scale. For movable points and baselines, one can just leave this field blank and it will allow these to be used on all scales, and thus for all gridded data sources.

## The Data Field Table

The file dataFieldTable.txt is where raw data variables that can be read directly from netCDF files are made known to the system. Each line in the file contains information about one raw data variable. Each line contains three or four text fields separated by vertical bars:

1. Variable name, which is a unique identifier for this raw data variable that is referred to in other tables.
2. netCDF ID, which is the name that this variable is given in netCDF files.
3. Now obsolete.
4. Special handling flag. For most variables this is just left blank. For variables that are ingested with units of pascals, a 1 is entered here as a signal to the derived field calculation software to do a pascals to millibars conversion, because all of the calculation software is set up to use millibars. A 2 is a signal that this is topography, and a 3 is a signal that this is the Coriolis parameter, neither of which comes from the netCDF files and so is generated internally by the GridSliceAccessor.

Without making any entries, the software that reads the data field table will generate a constant data field called `_ft`, which will return the forecast time in seconds, a constant data field called `_dt`, which will return the intrinsic frequency of the data (in seconds) as entered in the grid source table, and several constant data fields called `_dtX`, where X is the literal number entered in the

grid source table for the data frequency. These are meant mostly to be used to make decisions about which functions to invoke.

## The Virtual Field Table

The file `virtualFieldTable.txt` is where gridded data variables are registered for use in the volume browser. Included in `virtualFieldTable.txt` are instructions on how to make a variable if it is not a raw data grid that can be read directly from a netCDF file.

`virtualFieldTable.txt` makes much use of a feature in the low level module used to read in the text, namely that a `\` at the end of a line is a line continuation. This is because the entry for a variable must all be on one line, and by the time you add function specifications it is easy for a line to get longer than manageable. As such it may at times be useful to run the command

```
textBufferTest virtualFieldTable.txt > tempFile
```

This will yield a file where continuations are resolved and comments have been eliminated; thus it can be directly compared to the line numbers that are referred to in diagnostics from `testGridKeyServer`. The program `textBufferTest` builds in `$FXA_HOME/src/util` and should be in `$FXA_HOME/bin` at a field installation.

New virtual fields should always be added to the end of `virtualFieldTable.txt`; otherwise, the field indexing will change, which can break things.

Field users should place just their new virtual fields in a version of `virtualFieldTable.txt` that sits in their site-specific directory (`localization/LLL/LLL-virtualFieldTable.txt`) or their customization directory (`$FXA_CUSTOM_FILES/virtualFieldTable.txt`). In order to fix the indexing, such a file should begin with a line containing just a number; 300 is a good choice.

There must be at least seven primary text fields on each line, separated by vertical bars. In addition, there may be any number of function definitions, each of which has at least two primary text fields separated by vertical bars. The basic format of an entry for a single variable in the `virtualFieldTable.txt` file is as follows:

```
`varId'| CS | N? |`varName'|`units'|`displayTypes'|`planeList'| \  
  *`functionName',`planeList'|`varId',`plane'|`varId',`plane'|const| \  
  *`functionName',`planeList',`source'|`varId',`plane',`source'| \  
  *`functionName',`planeList',dTime|`varId',`plane'|`offset'|const| \  
                                `varId',`plane'|`offset'
```

In this basic format example, the function descriptions are just several examples of function descriptions that may or may not be present; a variable actually does not have to have any function descriptions.

Here is a description of the seven primary text fields on the first line of this format example:

1. The variable ID (`varId`) is the basic identification by which gridded data variables are known. In the file `dataFieldTable.txt` (described in the previous section), the first entry on each line comprises a list of all the variables that one might possibly read directly from the gridded data files. For a variable in that list, it is meaningful to make an entry with no functions. It is possible to define a variable with a variable ID not in that list, but it must have at least one function specification to be meaningful.
2. This text field contains information about how to interpolate this data to a cross section. If blank, it is assumed that it is inappropriate to interpolate this data to a cross section (e.g., for MSL pressure). Otherwise, this should contain the number of additional rows of horizontal grid points on each side needed to accommodate finite differencing for this field, beyond the minimum four points required for horizontal bilinear interpolation. Also, an additional optional comma-delimited number may be present, which refers to the enumeration `VectorRotation` in the source code file `GridMiscTypes.H`. This specifies the type of rotation to apply to vector data before displaying it. A [following section](#) lists these rotation types.
3. If a single `N` appears in this text field, the plane name will not be attached to the source and variable names when making a description of a single displayable gridded product of this variable type. Used most often with things like precipitation.
4. The variable name (`varName`) is the name that will appear in browser lists and in display legends for gridded products of this variable type.
5. The default units (`units`) is the units string that will appear in browser lists and in display legends for gridded products of this variable type. This may later be overridden by style information entries (see [styleRules](#)).
6. The display types list (`displayTypes`) is a list of possible ways to display this variable. The possible display types are `CONTOUR`, `ICON`, `IMAGE`, `BARB`, `STREAMLINE`, `ARROW`, `DUALARROW`, and `OTHER`. Each item in the list should be separated by commas. `CONTOUR`, `ICON`, and `IMAGE` are the ways in which one can display scalar variables. `BARB`, `STREAMLINE`, and `ARROW` are the ways in which one can display vector variables. `DUALARROW` really makes sense only for deformation vectors. `OTHER` is currently used for gridded soundings and time series plots. If a variable has no display types, then it is not displayable, but can be used as input to further calculations.
7. The plane list (`planeList`) is a list of planes for which this variable might be displayed. If empty, it can potentially be displayed for all known planes. In practice, all variables have some planes on which they are not available. A variable can still be used in calculations for planes on which it is not usable for display.

The rules for making a plane list will be described here separately, because they are quite involved and can apply either to a whole variable or to a function. To see a list of possible plane names, one really needs to run the program `testGridKeyServer` with a single ``p'` as an argument. This will direct a list of all known planes to standard output. The plane index will be the first item on each line, and the plane name will be the second item on each line. The simplest plane list is just a list of plane names separated by commas. One can also use some special syntax with the greater than symbol for identifying a range of planes. For example, the text `850MB>700MB` refers to all known planes with `MB` as their vertical coordinate and pressure values ranging from 850 to 700. One might also say `1000MB-500MB > 700MB-500MB`, but here what one gets is all of the composite (two-level) planes with `MB` as their vertical coordinate whose average vertical

coordinate value ranges from that of the 1000MB-500MB plane to that of the 700MB-500MB plane. Using optional grouping indices with planes can further affect this ordering. If two planes referred to in such a manner have either dissimilar plane types (standard or composite) or dissimilar vertical coordinate types, what you select is not well defined, so this is not recommended. It is also possible to use a single S, C, B, X, T, or D to specify all standard, composite, binary, spatial cross section, time-height cross section, or thermodynamic diagram planes, respectively. Finally, one can specify all levels of a certain vertical coordinate type by just putting that vertical coordinate type in the list. As an example, the plane list `S,MB,Surface' would result in selecting all standard planes that also had MB (pressure) as their vertical coordinate, plus SFC. `800MB>1000MB,BLyr' would select all pressure levels from 800 through 1000 millibars, plus the boundary layer. In addition, it is possible to use the ! symbol as a negation. For example, `S,MB,!50MB>150MB' would refer to all standard pressure levels except the ones from 50 to 150 mb, and `!Surface' would refer to all planes except the Surface.

This is a good place to introduce the concept of the volume plane. For every data variable for which there is sufficient information in three dimensions, and there is a meaningful entry in the second column (CS), a three dimensional grid will be created. This three dimensional grid is defined as existing in the volume plane, which has the plane name of `3D'. No data are displayed directly from the 3D plane; rather, 3D grids are inputs to making spatial cross sections, time-height cross sections, and gridded soundings.

## Functions

Here again are some idealized function descriptions:

```
*`functionName',`planeList'|`varId',`plane'|`varId',`plane'|`const'| \
*`functionName',`planeList',`source'|`varId',`plane',`source'| \
*`functionName',`planeList',`= `dataVar'|`varId',`plane',`source'| \
*`functionName',`planeList',`!= `dataVar'|`varId',`plane',`source'| \
*`functionName',`planeList',dTime|`varId',`plane'|`timeOffset'|const| \
    `varId',`plane'|`timeOffset'| \
*`functionName',`planeList',fTime|`varId',`plane'|`fcstOffset'|const| \
    `varId',`plane'|`fcstOffset'
```

The asterisk is a special signal to the software that parses virtualFieldTable.txt that this is a function entry. A given function entry starts at the asterisk and ends at the end of the line or at the next asterisk. The enumeration GridFunction in the source code file \$FXA\_HOME/src/dm/grid/gridEnum.h contains a list of all available functions (the leading `f\_' is not considered part of the function name). One can attach additional qualifiers to a function name, delimited by commas, to modify the behavior of a function entry. By attaching a plane list to a function name, one can make a particular function valid for only a certain list of planes for that variable. By attaching a source name to a function name, one can make a particular function valid only for that source. The literal string "dTime" causes the code to expect each variable ID to be followed by a time offset in seconds. The literal string "fTime" causes the code to expect each variable ID to be followed by a forecast time offset in seconds. A qualifier that begins with an equals sign is the name of a data variable that must be present in the source for the function entry to be used. A qualifier that begins with the string `!=' is the name of a data variable that must not be present in the source for the function entry to be used. Data variables listed in this

manner can be wild carded with an asterisk at the beginning or end of the string. These data variables are tested in the order given. If there are two equals signs (== or !=), additional data variables will be tested if the test fails, while a single equals sign means additional data variables will be tested if the test succeeds. The state of the last test performed determines whether testing against data fields tests OK in total.

A table of available functions follows, along with a short description of how each function is used. One can also gain a great deal of insight about how to use functions by looking at existing examples in virtualFieldTable.txt.

Each argument to a function is generally a variable ID, followed by an optional plane, with a few exceptions. Any text with a period in it is interpreted as a real constant; this notation is also used for time and forecast offsets. Constants can be used interchangeably with variables; where required, a single constant will be expanded into a grid of that constant, making it suitable for input to calculation routines that require a grid as an input. One can also just put in a vertical coordinate type, such as MB, which means create a constant field containing the vertical coordinate value of the specified level. If a particular function has a vertical coordinate type as an argument, then that function becomes usable only for planes with that same vertical coordinate type. Also, it is not necessary that the inputs to a function be grids that can be read directly from a netCDF file; they can be the results of other functions.

If no plane is provided, then the default plane is assumed. This just means the same level for which something is being calculated. If a plane is provided, it needs to be either a simple plane name (no ranges), the string "lower", or the string "upper". For composite planes, "lower" means the lower plane that makes up the composite plane, and "upper" means the upper plane that makes up the composite plane. For standard planes, "lower" means the next plane below that has data for the source and has the same vertical coordinate type, and "upper" means the next plane above that has data for the source. "upper" and "lower" have no meaning for planes with a non-parametric level type.

The order of the functions is important. The first function in the list which meets all the conditions specified by the qualifiers and for which all of the inputs are available is the one that is used. It is possible that, for a given variable, one function might be used for one plane or source, and a different function for another, depending on which of the possible prerequisite grids were available at each level for that source. Of course, one can force this behavior by attaching plane lists, source qualifiers, and/or data field qualifiers to the functions. In addition, there is a particular order in which variables have their functions resolved. First, an attempt will be made to resolve functions for variables that also appear in the data field table. Next, an attempt will be made to resolve functions for variables that have only one function entry. Thereafter, variables with an increasing number of function entries will become available to have their functions resolved, until an attempt has been made to resolve functions for all variables.

There are several functions that are handled in an exceptional manner. One is the function "MultiLoad." This function, as the name suggests, is used to build multi-loads, which can be used to auto-generate families. Variables with "MultiLoad" function always have their functions

resolved last. One should never try to use a variable defined with the "MultiLoad" function as an input to another variable.

Another exceptional function is the "Or" function. As the name suggests, its job is to return the first variable available from a list of variables. As such, when building function relationships, an "Or" function entry is considered usable when ANY of the variables is available, as opposed to the usual case of requiring ALL of the variables.

Finally, another exception is the function named "Import." This is a special function which allows one to bring data from one source into another. When using the Import function, an input variable ID must be qualified with both a plane and a source. If the geographic characteristics of the grid being imported are different from those of the source to which one is importing, a horizontal remap will automatically take place. Time interpolation will also be done if the two data sources have dissimilar data frequencies. Time interpolation can be suppressed if one puts an optional second constant argument in the variable list. The exception about "Import" is that if one makes an entry that says a variable is available to bring in from another source and that source is active, the software that builds the gridded data tables will believe you; it will not attempt to verify that an import can actually be done. This function should not be used to import vector components because it does not yet support grid-relative rotation.

## Available Functions

After each function name below is a notation that looks like I->O, for which I indicates the number of input arguments and O the number of output parameters; there may be multiple such entries. For the input argument count, n means any number one or greater, and for the output parameter count, n means the total number of input arguments. The total number of input arguments is not necessarily the same as the number of variables input to the function, because some of the input variables may return multiple parameters. Thus, the total number of input arguments is the sum of the number of output parameters from each input variable. Any mention of vectors refers to a two-dimensional horizontal vector.

Alias: n->n

Associate an arbitrary list of input arguments with a single variable.

Vector: 2->2 or 3->2

Associate arguments with a single variable, creating a vector. If two inputs, assume that these are just u and v components. If three inputs and the third input is a constant of magnitude 1 or 2, assume first two inputs are speed and direction. If third argument's magnitude is 1, assume degrees, otherwise radians. If positive, assume meteorological direction from, otherwise mathematical direction toward. In other cases with three inputs, the first two inputs are components that determine the direction, and the third is the speed to use.

Or: n->1

Return first in list for which any data actually exists.

Union: n->n

Return as many items as exist for the source. If one uses a Union function with only one input for a composite layer (e.g. 1000MB-500MB), it will compile a list of every

standard level available in that layer with the same level type. To use this feature, the composite layer needs to be assigned to the function, not the input variable.

Gather: 1->n

Return a list of all items that are the same except for a perturbation.

Either: n->1

When resolving functions, all inputs must be available to use the function. At run time, however, return first in list for which any data actually exists.

If: n->n-1

Associate an arbitrary list of input arguments with a single variable, except for the first argument. In other words, if the first variable is present, the rest become available.

Import: 1->1 or 2->1

Copy one parameter from another gridded data source, performing time and spatial interpolation as required. The optional second argument is always a constant and suppresses time interpolation if it exists.

Difference: 2->1 or 4->2

Perform scalar or vector subtraction.

Add: n->1 or n->2

Perform scalar or vector addition. All input arguments must be either vector or scalar; will produce garbage if mixed.

Average: n->1 or n->2

Calculate the arithmetic average of any number of arguments. All input arguments must be either vector or scalar; will produce garbage if mixed.

Multiply: n->1 or 3->2

Perform multiplication of any number of scalars or of a vector and a scalar.

Divide: 2->1 or 3->2

Divide a scalar by a scalar or a vector by a scalar.

LinTrans: n->1 or n->2

result = arg1\*arg2 + arg3\*arg4 + arg5...  
vecresult = vec1\*sca1 + vec2\*sca2 + vec3...

Max: n->1

Maximum value of each corresponding grid point. If a single 3D variable is passed in, will compute without considering the vertical coordinate information.

Min: n->1

Minimum value of each corresponding grid point. If a single 3D variable is passed in, will compute without considering the vertical coordinate information.

Exp: 1->1

Take the exponential of a scalar field.

Ln: 1->1

Take the natural log of a scalar field.

Power: 2->1

Raise first argument to the power of the second.

Poly: n->1

result = arg1\*arg2^arg3 + arg4\*arg5^arg6 + arg7\*arg8...

StdDev: n->1

Calculate the standard deviation of any number of input arguments.

Derivative: 4->1 or 6->2

Difference of the leading arguments divided by the difference of the last two arguments. Leading arguments can either be two scalars which yields a scalar result, or two vectors which yields a vector result.

Magnitude: 2->1

Calculate magnitude of a vector.

Dir: 2->1

Calculate direction from, degrees clockwise from north, of a vector.

Dot: 4->1

Calculate the dot product of two vectors.

Cross: 4->1

Calculate the cross product of two vectors.

Rotate: 3->2 or 6->2

Rotate vector in first two arguments by the number of degrees in the third, or transform the vector in first two arguments by the rotation matrix in the last four arguments. The arguments after the vector need to be constants.

CompBy: 4->1, 5->1, or 5->2

Returns the component of the first vector in the direction of the second vector. The optional last argument is a constant, which defaults to zero. The mod 1000 integer part is how many degrees to rotate the second vector before dotting it with the first. If the constant is not exactly an integer, then do not normalize the result by the magnitude of the second vector. If the thousands place is 1, just output the magnitude of the component. If the thousands place is 2, output a full vector, and if the thousands place is 3, output a full vector, the x component being the component along the second vector and the y component being the component along the k cross of the second.

Gradient: 1->2

Calculate the gradient of a scalar field.

NAdgdt: 3->2

Calculate the non-advective local change of the gradient of a conservative field. Inputs are wind components and the conservative field.

Laplacian: 1->1 or 2->2

Calculate the laplacian of either a scalar or vector.

Vorticity: 2->1 or 3->1

With a u and v component, calculate absolute vorticity. With third argument of a constant 0.0, calculate relative vorticity.

VortAdv: 2->1 or 3->1

With a u and v component, calculate absolute vorticity advection. With third argument of a constant 0.0, calculate relative vorticity advection.

Divergence: 2->1

Calculate horizontal divergence from u and v component.

Deformation: 2->1

Calculate total horizontal deformation from u and v component.

DefVectors: 2->2

Calculate horizontal deformation vector from u and v component.

Advection: 3->1 or 4->2

From u and v component and another parameter, calculate advection of that parameter. Parameter may be a vector.

DivParam: 3->1

From u and v component and another parameter, calculate divergence of that parameter.

GeoWind: 1->2

Calculate geostrophic wind vector from height.

PotVortMB: 8->1

Calculate potential vorticity based on data from two adjacent isobaric surfaces.

Arguments are upper theta, lower theta, upper pressure, lower pressure, upper u component, upper v component, lower u, and lower v.

PotVortK: 8->1

Calculate potential vorticity based on data from two adjacent isentropic surfaces.

Arguments are upper pressure, lower pressure, upper theta, lower theta, upper u component, upper v component, lower u, and lower v.

Temperature: 2->1 or 3->1

Calculate temperature from pressure and potential temperature or from pressure, virtual potential temperature, and specific humidity.

Theta: 2->1

Calculate potential temperature from pressure and temperature.

ThetaE: 3->1

Calculate equivalent potential temperature from pressure, temperature, and relative humidity.

VirT: 3->1

Calculate virtual temperature from pressure, temperature, and relative humidity.

Dewpoint: 2->1

Calculate dew point from temperature and relative humidity.

SpecHum: 2->1 or 3->1

Calculate specific humidity from pressure and vapor pressure or from pressure, temperature, and relative humidity.

MixRat: 3->1

Calculate mixing ratio from pressure, temperature, and relative humidity.

RelHum: 2->1 or 3->1

Calculate relative humidity from temperature and dew point or from pressure, temperature, and specific humidity.

CondPres: 3->1

Calculate condensation pressure from pressure, temperature, and relative humidity.

LiftedIndex: 4->1 or 5->1

Calculate lifted index from pressure, temperature, relative humidity, and 500mb temperature. Optional fifth argument is constant for arbitrary top pressure instead of usual 500mb.

Sweat: 4->1

Calculate SWEAT index from total totals index, 850mb dewpoint, 850mb wind, and 500mb wind. Last two arguments are vectors.

Heli: n->1

Inputs are a series of at least 3 vectors, the last being a storm motion and the rest being a stack of low level winds over which storm relative helicity will be calculated.

Cape: 5->1, 6->1, or n->1

Calculate Convective Available Potential Energy. The last four or five arguments are pressure, potential temperature, and specific humidity of the starting parcel; a constant flag for whether to use virtual (1) or plain (0) temperatures; and an optional argument (new in OB9) which is the upper pressure at which the CAPE computation terminates. If there are five or six input arguments, the first argument is 3D (virtual) temperature. Otherwise, the leading inputs are a list of pressure values and a list of corresponding (virtual) temperature values.

Cin: 5->1 or n->1

Same inputs as Cape, but calculate Convective Inhibition.

Dcape: 8->1

Calculate Downdraft Convective Available Potential Energy. The first two arguments are 3D temperature and dewpoint. The next three arguments are pressure, potential temperature, and specific humidity of the surface parcel. The last three arguments are constants which control how the calculation is done. The first constant is the maximum amount of liquid water available to evaporate into the parcel as it descends, in g/g. The second constant is the desired maximum RH of the descending parcel as it reaches the surface. The final constant is flag for whether to use virtual (1) or plain (0) temperatures.

WetBulb: 3->1

Calculate wet-bulb temperature from pressure, temperature, and relative humidity.

LapseRate: 4->1

Calculate lapse rate based on data from two adjacent quasi-horizontal surfaces.

Arguments are lower temperature, lower pressure, upper temperature, and upper pressure.

Hgt2Pres: 1->1

Calculate pressure from height based on standard atmosphere.

Mslp2Thk: 2->1

Calculate 1000-500mb thickness from MSLP and 500 height.

Alt2Pres: 2->1

Calculate surface pressure from altimeter setting and elevation.

TiltHgt: n->1

Compute the MSL height of a scan for the current home radar. First argument is a constant, the tilt angle in degrees of the desired scan. Any optional arguments that follow are not used in the computation, but only impose a non-static inventory on the result.

Shear: n->1

Compute the cumulative shear of a column of wind vectors. The first argument is always a constant. If it is null (>1e36), it is assumed that what follows is a stack of triplets: u, v, and a vertical coordinate value. If the leading constant is not null, then it is assumed that what follows is a stack of u and v, and the leading constant is the depth of the layer.

Slice: 2->1, 3->1, or 4->1

Define a quasi horizontal 2D surface based on the value of a parameter, and vertically interpolate data onto that surface. This is how theta-on-the-fly is implemented. Normally, the first entry is a constant that defines the vertical location of the surface, the second entry is the parameter that defines the vertical location of the surface (with a 3D plane qualifier) and the third entry is a 3D parameter to interpolate onto that quasi horizontal 2D surface. The vertical location of the surface is then where in the column the initial constant equals the value of the second entry. If the third entry is not present, it will default to pressure.

If the constant that defines the vertical location is  $1e35$  (or  $-1e35$ ), then the maximum (minimum) value in the column of the second entry is chosen to define the vertical position of the quasi horizontal 2D surface.

If a leading optional constant  $>1e36$  or  $<-1e36$  is present, this allows further control over how the slice is done. If this constant is  $>0$ , then the highest occurrence of the search value is used rather than the default behavior of using the lowest. If the magnitude of this constant is  $\leq 1e37$ , then the default vertically interpolated slice is used. If the magnitude is  $2e37$ , then the value of the nearest gridpoint vertically is copied to the output grid, and if the magnitude is  $3e37$ , vertical interpolation will be used if possible, but if not then the nearest gridpoint is copied.

Filter: 3->1

Apply a spatial filter to the first argument. The second argument is a constant which determines the distance over which to apply the filter, in km if positive, in number of grid points if negative. The third argument is a constant specifying how many times to apply the filter.

Test: 5->1 , 9->1 , 13->1 ...

This function takes five arguments at a minimum. There is an initial input argument, followed by a group of four arguments which can cause the input argument to be modified. There can be any number of these groups, each of which modifies the result of the previous argument group. For a single argument group, the basic functionality is that a constant operation type in the first argument determines how values in the input argument are tested against those in the second and third arguments; when the test is passed, values in the input argument are replaced with values in the fourth argument. When the first argument is positive, then the test is passed when values in the input argument fall within the range of values of the second and third. If it is negative, the test is passed if values in the input argument fall outside the range of values of the second and third. When the magnitude of the first argument is one, then testing and replacing is done on each individual corresponding grid point. When the magnitude of the first argument is two, the entire input grid is tested point by point and if ANY test is passed then the entire grid is replaced. If the magnitude of the first argument is three, then the entire grid is tested point by point and if EVERY test is passed then the entire grid is replaced. Also, non-zero digits in the thousands place of the operation type cause the argument to be used in an operation on the input data rather than just replace it. 1000 means add, 2000 means subtract, 3000 means multiply, and 4000 means divide. CAUTION: the user needs to be aware that once a particular value in a grid has been altered by a test, all memory of its original value is lost for the purposes of future tests. This can mean that two tests that work fine independently can fail to do what the user intended if strung together without forethought.

Accum: 2->1 or 3->2

Do an addition over time. First argument is a variable for which to add up the value, second argument is a floating constant which controls the time period over which values are accumulated. If the first argument is a vector, the output will be a vector. The time period argument is converted to the nearest integer before being interpreted; if the input argument is exactly an integer, then parts of an accumulation can be missing and a display will be made; otherwise the code will insist that all parts of an accumulation be present to make a display. If the integerized time period argument is zero, this means

accumulate forever. If it is greater than 99, this is the number of seconds over which to accumulate. If less than zero, then it is the number of time steps over which to accumulate. Otherwise, it is the forecast time step since which to accumulate (1 meaning the first step, usually the analysis).

Mean: 2->1 or 3->2

Same as Accum, but does an average.

MultiLoad: n->1

Create multi-loads, which among other things can be used to auto-generate families. The inputs to a multi-load function come in pairs, each pair corresponding to a single overlay. Each pair contains a floating constant followed by the variable to display in the overlay. The constant is converted to the nearest integer before being interpreted. A non-zero value in the ones place means this overlay should be toggled on by default. The tens digit is the display type to use: 0=contour, 1=icons, 2=image, 3=barbs, 4=streamlines, 5=arrows, 6=dualarrows, 7=other. A non-zero value in the hundreds digit means start a new pane. The thousands place is number of frames to load; 0 means the same as the number of forecast times and 99 means whatever the display is currently set for. If the legend ends in the word Family, the localization will attempt to post it to the main menu as a family. A list of constants after the last variable is the list of scale indices for which the multi-load is valid; if this is not provided, the multi-load will be generated for whatever scales are listed as valid for the source. A negative value here will be interpreted as the default density to use for this multi-load.

The functions `LvlQvec`, `LyrQvec`, `LvlFgen`, `LyrFgen`, `FnDiverg`, and `FsDiverg`, though fully implemented in the virtual field table, are complex and not described here. The function `Volume` is meant to be used internally in the code to put together three dimensional grids.

## Vector Rotation Modes

Here is the enumeration that refers to the vector rotation modes. For the most part, these are fully implemented. However, one could conceivably use this information, for example, to create displays for the components of a smoothed wind field.

```
enum VectorRotation {VR_NO_ROTATION = 0,
                    VR_EARTH_COORDINATES = 1,
                    VR_SECTION_COORDINATES = 2,
                    VR_COMPONENT_INT0 = 3,
                    VR_COMPONENT_ALONG = 4,
                    VR_GEO_MOMENTUM = 5,
                    VR_VERT_CIRC = 6};
```

---

**Author: Jim Ramer**  
**Last update: 28 Mar 08**

# Contouring arbitrary LDAD variables in AWIPS

## Introduction

The LDAD system in AWIPS can bring in an extremely varied set of hydrometeorological variables. In setting up the default AWIPS configuration, it is impossible to anticipate everything LDAD might bring in and set up display capabilities for it. This was the primary impetus for creating the adaptive plan view plotting subsystem within AWIPS. This capability allows infinitely configurable station model based plan view plots of any LDAD variable. It did not, however, allow the user to contour any of these variables.

The LAPS and MSAS components of AWIPS allow for the ingest and analysis of many standardized LDAD variables, and these can of course be contoured through the volume browser. However, LAPS and MSAS were never designed to be locally configurable for the addition of unique LDAD variables. As such, a configurable capability for analyzing and contouring an arbitrary point data variable on the fly was deemed necessary.

## Algorithm

The analysis package used to support such a capability should be viewed as more of a visualization tool than an objective analysis. As such, the most important thing is to match the observations as closely as is possible given the resolution of the grid used. That is, it would ideally result in contours that look like a person did a hand analysis. Since it runs in real time, it must also be fast.

The simplest well known and documented analysis scheme is the Barnes analysis. One feature of the Barnes is that one must pick a single distance scale representative of the data set. In areas where the distance scale is large compared to the observational density, details get washed out. In areas where the distance scale is small compared to the observational density, there is a tendency to create bullseyes. Add to this a tendency to approach the mean in large voids, and the Barnes is just not suited for matching the observations in a situation where the spatial characteristics of the data set are hard to predict. More complex schemes such as OI would have performance problems.

Here is a high level description of the analysis package that was coded up. First, all observations are resolved to the nearest grid point. A work grid of the same spatial resolution and expanded to twice the size of the analysis grid is used for this task to more accurately handle observations just outside the analysis grid. Where more than one observation resolves to the same point, all of these are averaged. As the values are assigned to the remaining analysis grid points, one always finds the unassigned grid point furthest from any assigned point. Once a value is assigned for that grid point, it becomes a virtual observation for subsequent calculations. When assigning a value for an analysis grid point, the space around the point is divided into eight octants. The nearest assigned point is found within each octant. If the point is mostly surrounded by data, each

point is weighted by the inverse of its distance to the fourth power. If only points off to one side are available the weighting is according to the inverse of its distance squared.

This scheme is fast and does an excellent job of adapting to the spatial characteristics of the data, even if they vary widely over different parts of the same analysis grid. The spectral response is not pleasing, in that there is a lot of 2-delta noise. However, that is the trade-off for having the analysis rigorously match the observations, which can vary over that space scale as well. Certainly, one would never want to hand the output of this analysis scheme to a numerical model. Nor is there any quality control, so caveat emptor.

## Sample Demonstration

There are demonstration versions of this for METARs in 5.2.2 with the menu entries left out. One can temporarily add entries to the end of the METAR->Other Plots submenu in nationalData/dataMenus.txt on the Surface menu to activate these. If one has an override file for dataMenus.txt, one would need to add these to the override file and run the -tables localization task. Here are what these entries need to look like:

```
productButton: 120120 # T
productButton: 120124 # T img
productButton: 120121 # Td
productButton: 120122 # Wind
```

To see these menu entries one needs to restart D-2D. This capability uses design files to gather the data and potentially do calculations on the data before it is handed over to the analysis. To understand more about how design files work, one should see [adaptivePlanViewPlotting](#). Grid depictables are given keys that point to these design files and to the point data set being used. When a properly formatted depict key entry is made, a PlotDesignData object is constructed and is asked for an analysis of the variable of interest. The Grid depictable then displays that data as it were any other grid.

We will use these demonstration entries to illustrate how one would go about setting up the contouring of an arbitrary LDAD variable. For any use of this capability, it is up to the user to choose unused keys for the implementation. For now, picking additional keys in the 120200-120999 range should be totally safe.

If one wanted to add menu entries for LDAD displays specific to the local site they would go into /data/fxa/customFiles/LLL-ldadMenus.txt. The product buttons in the demonstration menu are linked to displayable products using these corresponding entries in the file nationalData/productButtonInfo.txt.

120120		120120		Temperature		Metar Temp Analysis		0
120124		120124		Temp Image		Metar Temp Image		0
120121		120121		Dewpoint		Metar Dewp Analysis		0
120122		120122		Wind		Metar Wind Analysis		0

The entries for the local site would go in /data/fxa/customFiles/LLL-localProductButtons.txt. The header documentation at the top of the the default productButtonInfo.txt files explains how these entries work. Depict keys and product buttons are the same by convention, not by

necessity; it usually makes it easier to keep track of things. Here are the depict key entries for the demonstration, from nationalData/depictInfo.manual:

```
120120 | 3 | 82,120120,1003 | | 0|1 |METAR Temp Cont (F) |METAR T |1 |0 |1 |
|p,150,150,Tf |900
120124 | 10|82,120120,1003 | | 0|0 |METAR Temp Img (F) |METAR T |8 |0 |1 |29
|p,150,150,Tf |900
120121 | 3 | 82,120120,1003 | | 0|1 |METAR Dewpt Cont (F)|METAR Td |1 |0 |1 |
|p,150,150,Tdf |900
120122 | 4 | 82,120120,1003 | | 0|1 |METAR Wind (kts) |METAR Wind |1 |0 |1 |
|p,150,150,uW,vW |900
```

The entries for the local site would go in /data/fxa/customFiles/LLL-localDepictKeys.txt. These entries are probably the most difficult to get right and the most specific to the implementation, so even though header documentation exists in nationalData/depictInfo.manual, we will describe the individual entries here in detail. It will still be useful to be familiar with this header documentation. The first field is the depict key associated with the display, and is what goes into the product button table. The second field is the depictable type, 3 for contour, 10 for an image depiction, and 4 for wind barbs. In the third field is a list of data keys to be used (there will be more on data key entries later). 82 is the key for decoded METAR data, 120120 is the key for the associated design file, and 1003 is the key for the METAR station list. The corresponding keys for LDAD would be 87, implementation dependent, and 1011. The next two fields will always be empty and 0 for this type of display, and the next field will be 1 for a graphic and 0 for an image. The next two fields are legend information for the display and for logging. The field immediately after the legend information will be 8 for images and 1 for graphics, and the next two fields will always be 0 and 1. The next field is the color table index and as such needs an entry only for an image. The second to last field is the so called 'extra info' field, which is what signals to the grid depictable that it should grab point data and contour it. The 'p' is the flag that invokes this feature, the next two numbers are the dimensions of the analysis grid, followed by the item\_ids in the design file for what is to be analyzed, one for a contour or image and two for a wind barb display. The last entry is a notification delay, which one might want to make smaller, say 300, for LDAD data.

Here is the data key entry for the demonstration, found in the file nationalData/dataInfo.manual.

```
120120 | | | | | | |metarAnalDesign|.txt|metar contour demo design file
```

The format is straightforward: this just makes data key 120120 point to the appropriate design file, and the choice of this key is arbitrary and implementation dependent. The file where the corresponding entry for a local LDAD display would go is /data/fxa/customFiles/LLL-localDataKeys.txt. The name of the design file of course must be unique, and for a local implementation would reside in /data/fxa/customFiles/. Design files need to have names like \*Design.txt. All the default design files are in nationalData/, as is metarAnalDesign.txt.

What follows is enough of the contents of metarAnalDesign.txt to understand how the temperature data is presented for analysis. As mentioned before, one can learn more about design files from [adaptivePlanViewPlotting](#).

```
time_step 3600
```

```
item_id KF1
constant = 1.8
```

```
item_id KF0
constant = -459.67
```

```
item_id TC
```

```

type float
dimension scalar
netcdf_id temperature
min_valid 200
max_valid 340

item_id T10
type float
dimension scalar
netcdf_id tempFromTenths
min_valid 200
max_valid 340

item_id T
function or
inputs T10 TC

item_id Tf
function lintrans
inputs T KF1 KF0
placement upper_left
method formatted
format %d

```

The netcdf\_id keyword is always followed by the exact name of some netCDF data variable. For METARs, one can look these up in nationalData/metar.cdl. For LDAD mesonet data, one can look in nationalData/LDADmesonet.cdl for the default set of mesonet variables, but of course the whole point of this is to allow one to see contours of unconventional variables. Thus one would have to look at the override file for LDADmesonet.cdl in that case. What happens for the METARs is that two kinds of decoded temperature in kelvins are available, temperature and tempFromTenths. The `or' function is used to combine these two such that if the more precise tempFromTenths is available it will be used, otherwise the standard METAR temperature will be used. Finally, the item\_id `Tf' is defined by doing a units conversion on the `T' item such that the data is in degrees F. Tf is also the string that appears in the depict info that says which item to contour. A units conversion is needed here because there is no way to associate style info with one of these displays as there is for gridded data displays from the Volume Browser. The value 3600 after the time\_step keyword means that in a loop there will be one display per hour. The keywords `placement,' `method,' and `format' are significant in two ways. Most importantly, the existence of the `placement' keyword tells the design file that data from this particular item should be made available to the outside world, which is necessary for this capability to work. Second, one could temporarily copy this file to localizationDataSets/LLL/metarStdDesign.txt and then (after a D-2D restart) use the standard METAR plot to see a plot of the data values that will be contoured, which can help with troubleshooting. If one were to do the same thing to troubleshoot an LDAD contour, one could temporarily copy the design file to localizationDataSets/LLL/ldadMeso15Design.txt to test this with the 15 minute LDAD plot.

---

**Author: Jim Ramer**  
**Last update: 20 Mar 02**

## mainScript

The usage of mainScript.csh is as follows:

```
mainScript.csh {h} {n} {f} {t} {v} {+task} {-task} {-task} {loc_id}
{ingest_id}
```

Note that all of the arguments are optional. However, if run with no arguments at all, or with a single 'h' argument, this usage message will be printed out.

If run with no task or localization identifier arguments, mainScript.csh will perform all default localization tasks with the localization identifier of \$FXA\_LOCAL\_SITE and an ingest localization identifier of \$FXA\_INGEST\_SITE. The 'loc\_id' argument allows one to specify the localization identifier on the command line, and the 'ingest\_id' argument allows one to specify the ingest localization identifier on the command line.

The 'v' option will cause it to echo individual commands executed in the subordinate scripts. The 't' option will cause it only to verify that the localization ID selected is valid and list the tasks that would be run.

The 'n' option must be used if one is changing the customization environment for an existing localization. This means when the value of the environment variables FXA\_CUSTOM\_FILES or FXA\_CUSTOM\_VERSION changes. The 'n' flag must also be used if one wants to rerun an existing localization using a different ingest site.

There is now some logic in localization that allows it to detect when certain files are up to date and thus avoid recreating those files. If the 'f' option is present, this logic is disabled.

Currently, the complete list of default task options is as follows:

```
dataSups scales clipSups tables text topo grids radar maps wwa station
```

The '+task' option means perform that task and any tasks that follow. The '-task' option means just perform that task. One should use the +task option only once and it should be the first task option. One can use as many -task options as needed. Using the option '+dataSups' would be the same as the default behavior. To just verify whether a localization is viable, one can use a -task option for a non-existent task, such as '-x'. There are also seven non-default tasks called 'laps', 'msas', 'dirs', 'auxFiles', 'scan', 'purge', and 'trigger'. The 'laps' task is used to create metadata specifically for running the Local Analysis and Prediction System. Likewise, the 'msas' task creates metadata for running the MAPS Surface Assimilation System (MSAS). The 'dirs' task will assure the creation of all data directories on \$FXA\_DATA, as determined by the current state of the dataInfo.txt file, with all of its include files. The 'auxFiles' will create any other miscellaneous files that need to be moved to the data device. The 'scan' task creates metadata for running SCAN/FFMP. 'trigger' creates text product triggers. The 'purge' task will build the purge tables for the new purger. A task option of '-all' will

cause all default and non-default tasks to run. Additionally, the arguments ``-WS'`, ``-DS'` (or ``-DX'`), and ``-PX'` will result in running only those tasks absolutely necessary for localizations that reside on the workstation, data server, and application server, respectively. The argument `-WWA` will run just enough localization tasks to support warnGen full service backup.

---

**Author: Jim Ramer**  
**Last update: 17 Sep 04**

# Purging in AWIPS

The AWIPS intelligent data purger was phased in starting with OB5. The initial implementation included METARs, radar, and grids. OB6 added purging of satellite images and redbook graphics, with the rest of the datasets moved over in OB7.2. The purger attempts to use default purge parameters if necessary to purge any valid data key that points to a directory not in its tables.

The purger is a persistent process. Instead of waking up on a schedule and generating a huge burst of activity every so often, this purger is designed to maintain a very constant CPU load, and it informs the notificationServer of what it purges, which helps with notificationServer performance. The purger is smart enough to ignore directories and non-time-stamped files in normal purge operations, and has a separate mechanism for cleaning up non-time-stamped files. Files named literally 'template' will always be ignored by the non-time-stamped file logic; thus, no extra steps are required to manage template files. The purger allows for sophisticated purging schemes beyond simple version purging, among them time purging.

The data structures that control the purger are similar to the key tables that manage other aspects of AWIPS, and are overridable through localization in similar ways.

## The `purgeProcess` executable

The name of the executable that runs the purging is **purgeProcess** and it lives in the directory `/awips/fxa/bin/`. By default, this process runs on `dx1`, and it logs to the standard time-stamped directory in `/data/logs/fxa` as do all ingest processes. It can be stopped by script **stopPurgeProcess** and started by **startPurgeProcess** on `dx1` as `fxa`.

Note that if you stop the purger, a cron entry will restart it within ten minutes. It's generally not a good idea to let the system run without a purger; large inventories can cause slow-downs, and disk space can vanish relatively quickly. Also, do not manually start `purgeProcess`. Be sure to use **startPurgeProcess** when needed; it will prevent multiple `purgeProcesses` from running at the same time.

An important support file for `purgeProcess` is `/awips/fxa/data/purgeDataInfo.txt`. This file allows `purgeProcess` to see all data keys except for volume browser keys, and it should never be changed or overridden. The complete usage documentation for `purgeProcess` is found in [Appendix 2](#).

If EVENT logging is turned on for `purgeProcess`, performance summaries will be logged for each pass through the purge and data keys. Also, EVENT logging will result in a 'heartbeat;' there will be some log file activity every few seconds. VERBOSE logging will additionally log every time the process sleeps and will log every file that is deleted or would be deleted if the process was started without the `-commit` argument. DEBUG logging will add logging for each key that the process attempts to purge.

## Data structures for the purger

The main file that controls the behavior of the purger is nationalData/purgeInfo.txt. It is a standard AWIPS keyed access file: primary fields separated by vertical bars and sub fields separated by commas, with the key in the first field. Upon viewing the file, you will see that it starts with a large amount of user documentation, which is included here in [Appendix 1](#). The rest of the file contains this:

```
60 | | | 2- | 288 // aiv/ncwf/netcdf
72 | | | 2- | 72 // aiv/convSIGMET/netcdf
62 | | | 14- | ,14- // point/LSR/netCDF
.
.
.
#include "radarPurgeInfo.txt"
#include "satPurgeInfo.txt"
#include "redbookPurgeInfo.txt"
#include "gridPurgeInfo.txt"
#include "localPurgeInfo.txt"
```

Notice that there are many specific purge entries in the file plus several #include lines. There are default implementations of all of the include files except for localPurgeInfo.txt. As the name would suggest, localPurgeInfo.txt is meant to bring in purging info for locally defined data sets. One can also change (override) the default purging behavior for individual data sets in localPurgeInfo.txt.

There is a localization task, -purge, whose job it is to manage the purging tables. Running it will take all purging information in the system, including defaults and local overrides, and move them into the localizationDataSets/LLL/ directory for the site.

## Understanding individual purge table entries

A great deal of understanding of how these entries work can be obtained by reading the header documentation from purgeInfo.txt, reproduced in [Appendix 1](#). Here we will dissect some existing entries in the default system as an aid to this understanding. Here is the entry for METAR purging:

```
82 | | | 2- | 34 | 38,,=3: | 42,,=6: | 50,,=24:,+12: // point/metar/netcdf
```

The key '82' is what means this is for purging METAR files. This is the same data key as exists for METARs in nationalData/dataInfo.manual. The next field is blank because we are relying on the data key to specify the directory, and the next key is blank because we want no special behavior not specified in the main set of purge groups. The next field, which contains '2-', means remove any files that are not time stamped if they are older than two days. After that, we have four purge groups. In order for a file to be kept, it must be OK with at least one of the purge groups. Within a group, it must pass all tests specified. The first group represents a straight version purge to 34 versions. The second group means keep up to 38 files that are time stamped exactly at 3 hour intervals; this means 4 additional files (12 hours, in this case) beyond the 34 of the first rule. The third group means keep up to 42 files time stamped exactly at 6 hour intervals (four more files, or a day's worth), and the last group means keep up to 50 versions stamped exactly at 24 hour intervals, at 12Z (eight more days). Because the largest versions parameter is

50, the purging will reduce the set of kept files to no more than 50 when a purge operation is complete.

This "intelligent" purge strategy allows us to provide a much better user experience than would a straight version purge. In comparison to a 36-version purge, we are keeping only 50% more files, yet we are going back an extra day and a half with the 3/6 hour frequency, which makes the 24 hour change displays more useful, and we are going back a whole extra week with the files that contain the 24 hour precipitation.

If the default entry for METARs were as follows, the purging would be essentially the same:

```
888888 | point/metar/netcdf | | 2- | ,~34:00 | ,~38:00,=3:00 | ,~42:00,=6:00  
| ,~50:00,=24:00,+12:00
```

In this entry, we have just picked an undefined key and entered the directory into the purge info. This will purge the same but the notificationServer will not learn about what is purged, as that requires actual data keys. The purge parameters are in hours instead of versions, but since METAR files are always hourly, this is equivalent.

Some additional examples worth viewing are from the default version of gridPurgeInfo.txt:

```
1069999999 | 1070000000,1070000255 | | 2- | 2 | ,~24:01  
301 | | ,,,i | | 10000 // Grid/SBN/Raw  
1070000019 | | ,1:00 | 2- | 2 // CONUS212/MesoEta
```

The first entry uses the arbitrary undefined key 1069999999 to carry purge info for actual data keys in the range from 1070000000 to 1070000255. Starting with OB5 there are specific data keys that point to the gridded data netCDF files as a whole, as opposed to individual volume browser displays, and this is the entire range of these keys (see localizationDataSets/LLL/gridNetcdfKeys.txt). This default entry for all gridded data netCDF files will keep any files up to two versions or up to a day old. Thus, data created daily would be kept for 2 versions, hourly files would be kept for 25 versions. The second entry is for purging the safe store directory (temporary storage for unprocessed SBN grids). The ',,,i' means we ignore time stamping and just keep the newest 10000 files. The last entry is specifically for the 40km NAM (historically, MesoEta) grids. The text at the end is just a comment; the real control of which data set is involved is via the key. We keep 2 versions, but the ',1:00' entry means wait until the newest file has had nothing written to it for an hour before removing the oldest file. This gives the newest run a chance to complete before the older run is removed.

### **File override resulting from running the -purge task**

All of the overridable files that control purging have their default version in nationalData/. The list of currently implemented files is purgeInfo.txt, gridPurgeInfo.txt, radarPurgeInfo.txt, radarPurgeInfo.template, redbookPurgeInfo.txt, and satPurgeInfo.txt. New for OB9, we are implementing a default version of dialPurgeInfo.template to purge the FAA radar imagery. The .txt files are subject to replacement override from realization files and to append override from site-specific files. The .template files are subject to replacement override from realization files

and to append override from both site-specific and customization files. The file localPurgeInfo.txt is for override only and is subject to append override from both site-specific and customization files. It is expected that the vast majority of locally-defined purge entries will be in customFiles/localPurgeInfo.txt. Purge information entered here for specific data keys will override that defined by default, plus one can add purging information here for any locally-defined data sets.

If one does override the default purge information, there are some things to keep in mind. Once your override files have been edited and the -purge task has been run on dx1, the purge process needs to be restarted. If the override is wanted long term, the -purge task should also be run on the backup host for the purgeProcess (currently dx2). If one makes entries where the key functions not as an actual data key but rather as an arbitrary purge key, one should not use a key that is already an actual data key, nor should the arbitrary key fall within the range of data keys provided. It is always OK to make an entry for an existing purge key in an override file, as long as the intent is to replace the existing default purging information associated with that purge key. For now, if local users need to make entries for arbitrary purge keys, it is suggested that the arbitrary purge key used be in the range 4000000000 to 4200000000.

If one makes a purge entry that has problems with one of the purge groups (ver,period,delta,round) but still has usable information for other purge groups, the purger will attempt to perform purging operations based on the groups it could successfully parse. A trailing vertical bar will be interpreted as an empty and therefore problematic group, but will not necessarily prevent purging operations from occurring. If none of the purge groups can be interpreted and there is no scour information, then that purge entry will not be used and a diagnostic like this will appear in the log file:

```
purgeProcess 15:03:05.339 DM_PurgeInfo.C 1388 PROBLEM: No usable purge info
for purge key: 123456 Num fields: 4
```

## Radar data purging

Here we discuss in more detail how the purge tables that control radar data are managed. The radarPurgeInfo.txt resulting from localization (found in dx1:/awips/fxa/data/localizationDataSets/LLL) is a combination of information from nationalData/radarPurgeInfo.txt (containing catch-all entries), radarPurgeInfo.template (for associated radars), and dialPurgeInfo.template (for FAA radars), plus any overrides.

If one looks in nationalData/radarPurgeInfo.txt, one sees three entries, to wit:

```
2200000000 | 1073741824,2147483647 | | 2- | 30 | 36,,~1:00
2200000001 | badRadar | ,,,i | | 60
//2200000002 | radar/raw | ,,,i | | 20000
```

The first is a default entry for the entire range of possible radar data keys, saying version purge to 30 versions, and keep an additional 6 files at approximately one hour intervals. (For RFCs, there is a realization version of this file that is the same except it purges to 60 versions plus an additional 12 at one hour intervals.) Next is a purge parameter of 60 files for the directory where

undecodable radar files are placed, and finally is a commented out entry for the safe store directory for radar. We currently do not purge the radar safe store but this is here just in case we need it.

The file `nationalData/radarPurgeInfo.template` contains purge entries for associated radars in terms of generic keys. During localization, these are converted to radar-specific keys and appended to the items above. The only current entry is a catch-all for all generic radar data keys, purging to 72 versions, with the last 12 hourly. If in the future we wish to purge certain products differently (suppose, for example, that we want to keep one-hour precip for 24 hours), that will be added here. To do the same for OCONUS radars, the likely approach will be to add site-specific override versions of `radarPurgeInfo.template` (`LLL-radarPurgeInfo.template`) to the release. If a site wanted a specific generic data key for associated radars to be purged differently, they would do this in the file `customFiles/radarPurgeInfo.template`.

The localization scripting also processes `dialPurgeInfo.template` exactly as `radarPurgeInfo.template` is processed, producing like purge entries for FAA radars. It is not used for non-associated radars because we rely on the main all-radars entry in `radarPurgeInfo.txt` for purging files from those radars.

Another special feature of the purger that relates to radar data is how it handles subdirectories named `inventory_parameters/`. In these subdirectories (for a few products including SRM and VAD) reside time-stamped text files corresponding to each of the data files in the top-level directory. These files contain additional information that gets presented with inventories in the D-2D user interface. The purger automatically purges any subdirectories named `inventory_parameters/` at the time it purges the top-level directory and with exactly the same purge parameters.

## Appendix 1) header documentation from `purgeInfo.txt`

```
//
// This file, purgeInfo.txt, is where purging information is entered for
// data keys. This file has #include lines for keys for radar, satellite,
// grids, and local keys. Each line in the file normally supplies the
purging
// information for one data key, but sometimes can refer to ranges of
// keys. Here is how these entries are formatted:
//
// key | dir | r,w,c,i,l | scourPer | ver,period,delta,round | ...
//
// key - A data access key to give purge info for.
// dir - If this is non-blank, then we assume that 'key' is not a real
// data key, but an arbitrary key used to drive the purging of this
// particular directory. If this is two comma delimited keys
// instead of a directory, then this is assumed to be a range
// of keys that this is the default purge info for. Five keys
// are `min,max,div,minRem,maxRem', which means the whole key
// must be between the first two, and the remainder when divided
// by div must be between the last two.
// r - A non blank entry means go through directories recursively.
// w - A non blank entry means purge all prefix/suffix combinations
```

```
//      found in the directory separately.
//  c - Time period to wait after the mod time of the latest file to
//      purge normally; this allows the most recent file to be completed
//      before the oldest is purged.
//  i - Non-blank entry means ignore time stamps, no delta times or round
//      times allowed in this case.
//  l - A non blank entry means do not actually purge by this entry, only
//      log what would have been purged.
//  scourPer - Any non time stamped files in the directory will be removed
//              if older than this length of time. Defaults to zero, which
//              means do not scour (see the time string definition below).
//              Any comma delimited entries after the scour period in this
//              field are assumed to be the names of files which will be
//              neither purged or scoured.
//  ver - Number of versions to keep; defaults to 0 which means do
//        not version purge.
//  period - Max period between the current time and oldest time stamp of
//            file to keep; defaults to 0 which means do not time purge.
//            A leading tilde (~) on the period means calculate from the
//            latest file time stamp instead of the current time.
//  delta - File with time stamp separated by less than this from next
//            newest file will not be kept; defaults to zero which means
//            do not consider time separation. If a leading equals (=),
//            only keep files an exact multiple of this delta time, if a
//            leading tilde (~) only keep the one file closest to an
//            exact multiple of this delta time.
//  round - Round times by this before deciding whether to purge; defaults
//            to zero which means do not round. The rounding time interacts
//            with the delta, but not the period. If a leading plus sign (+),
//            add the time instead of rounding by it. If consecutive files
//            round to the same time, then if one is kept they will all be
//            kept.
//
//  To be valid, an entry must have at least four vertical-bar-delimited
//  fields. There can be any number of ver,period,delta,round groups. If
//  the first possible ver,period,delta,round field begins with an alphabetic
//  character, that is assumed to be a command to run to handle purging.
//  At a minimum there must be a key and a usable ver,period,delta or scour
//  parameter.
//
//  Time lengths are encoded as dd-hh:mm:ss, where dd is days, hh is hours,
//  mm is minutes and ss is seconds. Could say `2-` for 2 days, `:30` for
//  thirty minutes, `:4:30` for four and one half minutes, `4:` for four
//  hours, or `1-18` for one and three quarter days (one day, 18 hours).
//
//  For each ver,period,delta,round group, a file must pass all
//  tests to be considered for retention. If there are multiple groups,
//  a file must only pass the tests for one group to be retained.
//  Since vb keys (>=0x80000000) never point specifically to a directory,
//  it will be assumed that keys in this range will always be used
//  for designating the default purging info for ranges of other data
//  keys. Key 4294967295 (0xFFFFFFFF) is the largest possible data key and
//  is reserved for defining the default purging info applied to any data
//  key for which no purging info is found. There are internal defaults for
//  this so an entry for this key is not mandatory.
//
//  Other predefined ranges of purge keys that do not correspond to
```

```
// existing data keys:
//
// 1069000001 - 1069000009 Satellite arbitrary purge keys.
// 1069999999 - 1070000255 Gridded data purging keys.
// 3800000000 - 3999999999 Site-specific arbitrary purge keys.
// 4000000000 - 4200000000 User-defined arbitrary purge keys on site.
//
```

## Appendix 2) Usage documentation for purgeProcess

```
purgeProcess {-commit} {-noall} {-logall} {-wait}
             {-one} {-two} {-three} {-four}
             {k key1 key2 ...} {opsPerSec} {+cycleTime}
```

-commit - literal, must be present for performing purge/scour operations, otherwise will just log files to remove.

-noall - If a non-VB data key has no purge entry that either matches it or includes it in its range of keys, do not purge it by default.

-logall - If a non-VB data key has no purge entry that either matches it or includes it in its range of keys, only log what would be purged.

-noinit - Begin purging operations immediately...don't surf all tables first to check for redundancies.

-wait - wait thirty seconds to start execution to allow a debugger to be hooked up.

-one ... - Number of passes to perform before stopping, by default will never stop until interrupted.

k - If literal flag `k' is present, will only test purging the listed keys, which can be purge keys or data keys. If a data key is given which relates to a purge rule with a different key, the purge rule key must come before the data key for the data key purge to work.

opsPerSec - is the number of directories to purge per second, defaults to 5.0.

cycleTime - Desired number of minutes to cycle through all purge keys and data keys, defaults to 30.

---

**Author: Jim Ramer**  
**Last update: 25 Apr 08**

# Radar Localization

## Introduction

In the OB3 release the general organization of radar tables changed a great deal, mostly in support of new Volume Coverage Patterns (VCPs) that came on line with many new tilt angles. The previous user interface, which was based on loading specific tilts, would have been stretched beyond its limits had we tried to use it for all the different tilts (about 40 in all) that became available. Thus, we switched to an interface that is based on ranges of tilts.

Furthermore, with the increased emphasis on 8 bit products, we changed the mechanism whereby data sets that are available in multiple bit depths/data levels are displayed. For base reflectivity, for example, it will try to load the 8 bit/256 level product first. If that is not available, an attempt will be made to load the 4 bit/16 level product, and then finally the 3 bit/8 level product. No matter which product is loaded, the data are always mapped to a common fixed dBZ to color mapping for display. This functionality required formalizing the way we make entries that describe the image value to data value mapping for 8 bit products. Unlike the legacy 4 and 3 bit products, 8 bit products do not contain this information in the header, so we have to enter it into tables.

We will begin by discussing the ramifications to localization caused by the VCP-driven table organization. Next we will discuss how to make certain changes to the output data value to color mapping based on the new data value mapping tables. Finally, we will include some general hints about radar localization.

## Effects of VCP-driven key reorganization

The general functioning of radar localization is that files containing generic radar product keys (nationalData/radar\*.template files) are converted into radar-specific keys for each radar in the site's radarsInUse.txt file and written to the localization data set as radar\*.txt files.

Switching from an organization based on specific tilts to one based on ranges of tilts meant at a minimum that some depict keys would go away, which necessitated patching procedures for OB3. Since procedures were being patched anyway, we took the opportunity to reorganize the radar generic key space to something more logical. If one is curious about how this was done, read the header documentation in the new radarDataKeys.template and radarDepictKeys.template files in nationalData/. The primary upshot of this reorganization was that any key-based radar override files that one had been using became invalid unless the generic key entries in them were changed. More on this in the general hints section.

## Modifying data value mapping

In AWIPS we use a data structure called an image style entry to describe how data are mapped from image counts to physical data values like dBZ. See the header documentation of nationalData/imageStyle.txt for more information about the format of style entries. As previously mentioned, the 4 and 3 bit products usually have information in their headers that can be used to

map from image counts to data values, but not so for most 8 bit products. Thus, the AWIPS radar subsystem uses image style entries to describe both the characteristics of raw 8 bit data and how data are displayed. Under the hood, AWIPS detects when the data style and display style are different, and will rescale the image data so that, for example, 40 dBZ is always 40 dBZ regardless of what the output display style looks like.

Previous releases had no clean separation between keys that were meant only to carry style info for 8 bit data sets and style entries that were meant for controlling how 8 bit image products were displayed. To rectify this, we have introduced a new file called `nationalData/radarGenericImageStyle.txt`. This file contains display style entries for mosaics (for more on mosaics see [radarMosaics](#)) and all the style entries that are meant to describe how image counts map to data values for eight bit products. In here are also entries that control how the image counts in 4 bit VIL data are mapped to approximated reflectivity for the `VIL/Comp Ref' product. The mosaic display style and the VIL to reflectivity mapping is fair game for user configuration, but the rest of the entries that describe the meaning of 8 bit data sets should not be changed unless those data sets change. These entries remain generic - the radar localization does not convert them to radar-specific keys.

Before 8 bit products came on line, very few radar products had display style different from that associated with the data set. Now that we are combining all bit depths for reflectivity and velocity into a single menu selector and data value to color mapping, having different display style entries is the rule rather than the exception for base reflectivity and velocity products. The display style entries for radar products are found in `nationalData/radarImageStyleInfo.template`, and these are based on generic depict keys. If an entry is changed in `radarImageStyleInfo.template`, one will also often need to change the units or color table entry in `radarDepictKeys.template`, or perhaps the value of one of the `RADAR_*` directives that control the color table values for classes of radar product (see [directives](#)).

An example of something one might change is the scaling for the 8 bit Vertically Integrated Liquid water product (DVL, key 50843). There are currently two commented-out entries for it in `radarImageStyleInfo.template`. This product is unique because the data has hybrid log/linear scaling, and furthermore that scaling will change slightly from ORPG build 8 to build 9. The first commented-out entry is the same as the default scaling of the data for build 8. Uncommenting it will have no effect other than to generate redundant style entries while build 8 is being used; when build 9 is put in place it will have the effect of forcing the color to VIL value relationship to remain what it was during build 8. The second commented-out entry is one that is the same as the four bit VIL data. If one were to uncomment that second entry and run the `-radars` localization task, one would end up with linearly scaled output for the digital VIL. One would probably also want to change the color table for this product in `radarDepictKeys.template` to be the same as the 4 bit VIL in this case. The point of this is not necessarily to recommend either of these changes, but merely to provide an example of how one might do this.

Another example of something one might want to change is the output scaling for the Storm Total Precipitation, key 50111. In places where rainfall intensity is highly seasonal, it might make sense to have a smaller range of values when heavy rainfall is less common and a larger

range of values when it is more common. Alternatively, one may want to convert to a pure log output scaling, which as mentioned before can now be done.

### General hints for radar localization

1. When customizing, the approach always needs to be 'start with the delivered defaults, and then edit in your changes;' *never* 'start with my old customization, then attempt to reimplement the new capabilities.' The second approach will inevitably end up negatively affecting some default capabilities. This is not theoretical - there have already been several instances where SST has had to spend time troubleshooting radar display problems resulting from this. This was *extremely* important for OB3 because the keyspace had been rearranged.
2. When you override nationalData/radar\*.template files, do not edit the files in place in nationalData/. You need to provide override files at either /data/fxa/customFiles/radar\*.template or localization/LLL/LLL-radar\*.template. Also, except for radarDataMenus.template where order is important, there is no reason to replicate every generic key entry. Enter only the ones you want to add or change, and the rest will retain their default entries from the nationalData/ file. Overriding every generic key makes restoring customizations after an upgrade more complicated, and so is strongly discouraged.
3. Your overrides for nationalData/radar\*.template files are usually better placed at /data/fxa/customFiles/radar\*.template instead of localization/LLL/LLL-radar\*.template. If you override radarDataMenus.template in customFiles/ and you want to change the whole layout, start the file with a line containing '#replace'; otherwise your entries will be added to the bottom of the menu. Occasionally you will want to be able to have a localization used for service backup have different radar tables than your primary localizaton, and for this you will want to use LLL-radar\*.template files. Thus, this is a recommendation, not a hard and fast rule.
4. It is usually better to change radar color tables using directives instead of directly overriding the radarDepictKeys.template files. See [directives](#) for more information, specifically, the directives RADAR\_Z through RADAR\_VH. Also, it is perfectly OK to enter user-defined color table indices here, as well as something like '1167,100' (table and default brightness). Not all depict keys have color table entries linked to one of these directives, and sometimes there will be legitimate reasons to assign a totally different color table to a key that is linked to a directive by default. Thus, again, this is a recommendation and not a hard and fast rule.

### Appendix -- Table of current tilt angle bins

primary angle	min angle	max angle	VCP 12	VCP 121	VCP 112	VCP 11	VCP 21	VCP 31	VCP 32
0.5	0.4	0.7	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.9	0.8	1.1	0.9	---	---	---	---	---	---
1.5	1.2	1.6	1.3	1.5	1.3	1.5	1.5	1.5	1.5
1.8	1.7	2.0	1.8	---	---	---	---	---	---
2.4	2.1	2.6	2.4	2.4	2.1	2.4	2.4	2.5	2.5
3.4	2.7	3.6	3.1	3.4	2.9	3.4	3.4	3.5	3.5

4.3	3.7	4.6	4.0	4.3	3.8	4.3	4.3	4.5	4.5
5.3	4.7	5.6	5.1	---	4.8	5.3	---	---	---
6.0	5.7	6.6	6.4	6.0	6.1	6.2	6.0	---	---
7.5	6.7	8.0	8.0	---	7.7	7.5	---	---	---
8.7	8.1	9.5	---	---	---	8.7	---	---	---
10.0	9.6	11.0	10.0	9.9	9.7	10.0	9.9	---	---
12.0	11.1	13.0	12.5	---	12.2	12.0	---	---	---
14.0	13.1	15.6	15.6	14.6	15.5	14.0	14.6	---	---
16.7	15.7	17.9	---	---	---	16.7	---	---	---
19.5	18.0	22.0	19.5	19.5	19.5	19.5	19.5	---	---

**Author: Jim Ramer**  
**Last update: 17 Oct 05**

## Radar Mosaics.

For build 5.1, the radar mosaics now work a little differently. First, the radars that go into the mosaic are no longer always automatically the set of radars that are in radarsOnMenu.txt. For an RFC or national center, this will still be the case if no action is taken by the site. However, for WFOs, the number of radars in any mosaic will be limited to the closest nine. Most importantly, however, is that there is now an option to supply a table that will control in a very specific manner which radars are in the mosaic, and will even allow mosaics of more than one set of radars.

If created on site, this table should be placed at the file path \$FXA\_CUSTOM\_FILES/mosaicInfo.txt. If placed under configuration control the path localization/LLL/LLL-mosaicInfo.txt should be used. Here is an example of such a table. Placing the comments in line is a good idea as it helps whoever might need to manage the table.

```
// This file controls each radar mosaic that can be generated.
// Each line represents one mosaic. Here is a model of each line:
//
// scales | file/list | count | center | title
//
// Here is the meaning of each column:
//
// scales : A space delimited list of scale indices to use for this mosaic.
//          Defaults to the contents of mosaicScales.txt.
// file/list : A space delimited list of radars to use in this mosaic, or
//             alternatively, a file where this list is. If not supplied
//             defaults to radarsInUse.txt.
// count : Max number of radars to include in the mosaic. If this is an RFC
//         or national center, this can be any number and will default to
//         all available radars. Otherwise, this will default to nine, and
//         will arbitrarily be limited to nine.
// center : Takes the radars in the list closest to this point, up to the
//         value in the `count' column. This is a lat/lon, which defaults
//         to the contents of CenterPoint.dat, which should be the center
//         of the area of responsibility.
// title : Should be unique for each line. A line without a title will
//         appear directly on the main `Radar' menu, others will be in
//         a pull right.
//
// If a version of this file is not supplied for the localization, a
// default version will be created with one entry that looks like this:
//
//      | radarsOnMenu.txt | | |
//
//
//
5 4 | kcys kftg kgld kgjx kpux | | |
3  | | | 8 | 40 -109 | West
3  | | | 8 | 35 -95  | South
```

In the example, the default Mosaic that has its menu entries directly on the top level menu called `Radar' has the five radars kcys, kftg, kgld, kgjx and kpux in it. Immediately following those menu entries will be a pull-right labeled `West'. In that pull-right will be menu entries for mosaics that include the 8 radars nearest to the point 40N 109W. Immediately following the pull-

right labeled `West' will be a pull-right labeled `South'. In that pull-right will be menu entries for mosaics that include the 8 radars nearest to the point 35N 95W.

The main Mosaic will be shown on the WFO and State scales (scales 4 and 5) and the other two will be shown on the regional scale. Note that if no scales are listed for the mosaic, the contents of mosaicScales.txt will determine the scales, and that file defaults to just the State scale if it is not provided.

The reader should note that unless a radar appears in the radarsInUse.txt file, it will not be included in a mosaic. The radarsInUse.txt file that applies is always whatever one is applicable for the localization running on the data server.

The remaining files in the default system that apply to mosaics all reside in the directory localization/nationalData/, and they are called mosaicDepictKeys.template, mosaicDataMenus.template and mosaicProductButtons.template. The reader is directed to documentation in the headers of those files for additional information about how those files work. If one needs to make local changes to these files, the user should create copies of the originals in the file paths LLL-mosaicDepictKeys.template, LLL-mosaicDataMenus.template, and LLL-mosaicProductButtons.template in the directory localization/LLL/. After successful edits have been completed, the new files should be saved off somewhere so that they can be restored after upgrades.

Once one has edited any of these files, one needs to run the -radar localization task on a workstation host and then restart the D-2D on that host in order to view the newly configured mosaics. One must run the -radar task and then restart the notification server on the appropriate machine in order to get notifications for the newly configured mosaics.

---

**Author: Jim Ramer**  
**Last update: 14 Aug 00**

## Readme Grids

The user should note that if one is not working in a source code environment (one that only contains the \$FXA\_HOME/data and \$FXA\_HOME/bin directories) all documentation and metadata files will be in \$FXA\_HOME/data, all utility programs will be in \$FXA\_HOME/bin, and all localization scripts will be in \$FXA\_HOME/data/localization/scripts/, and all scripts are set up to recognize which type environment you are in and will respond to this. Also, if one is not working in a source code environment, checking out and checking back in files does not apply.

In general, there are five steps in updating anything involving grids.

- 1) Check out any of the affected files. There are quite a few that might be affected by a change involving gridded data...here is the current list of all files that might affect the default system:

```
$FXA_HOME/src/localization/nationalData/gridSourceTable.template
$FXA_HOME/src/localization/nationalData/activeGridSources.txt
$FXA_HOME/src/localization/nationalData/gridPlaneTable.txt
$FXA_HOME/src/localization/nationalData/virtualFieldTable.txt
$FXA_HOME/src/localization/nationalData/contourStyle.rules
$FXA_HOME/src/localization/nationalData/gridImageStyle.rules
$FXA_HOME/src/localization/nationalData/iconStyle.rules
$FXA_HOME/src/localization/nationalData/arrowStyle.rules
$FXA_HOME/src/localization/nationalData/*.wc
$FXA_HOME/src/applications/volumeBrowser/browserFieldMenu.txt
$FXA_HOME/src/applications/volumeBrowser/browserPlanViewMenu.txt
$FXA_HOME/src/applications/volumeBrowser/browserSoundingFieldMenu.txt
$FXA_HOME/src/applications/volumeBrowser/browserSoundingMenu.txt
$FXA_HOME/src/applications/volumeBrowser/browserSpacePlanViewMenu.txt
$FXA_HOME/src/applications/volumeBrowser/browserSpaceXSectionMenu.txt
$FXA_HOME/src/applications/volumeBrowser/browserTimeHeightFieldMenu.txt
$FXA_HOME/src/applications/volumeBrowser/browserTimeHeightMenu.txt
$FXA_HOME/src/applications/volumeBrowser/browserXsectFieldMenu.txt
```

- 2) Edit required tables.

The file virtualFieldTable.txt is a table where one can control how to generate a particular field, or add new derived fields. The file gridSourceTable.template is where one introduces new gridded data sources to the system, and activeGridSources.txt is where you make sources displayable. The file gridPlaneTable.txt is where one adds new levels to the system. In all of these tables, new items should be added to the end, never inserted, because that changes the indexing. Documentation for how to manage these tables can be found in [gridTables.html](#).

The files contourStyle.rules, gridImageStyle.rules, iconStyle.rules, and arrowStyle.rules control the look and feel of renderings of gridded data. Documentation for how to manage these tables can be found in [styleRules.html](#).

The files browserFieldMenu.txt, browserTimeHeightFieldMenu.txt and browserXsectFieldMenu.txt contain, respectively, the menu layouts for plan view fields, time height fields, and cross section fields. The file browserPlanViewMenu.txt contains the menu layout for all of the plan view level selections. The rest of the browser menu layout files should rarely, if ever, be changed. These files all have documentation in the file. If not in a source code environment, the default versions of these reside in \$FXA\_HOME/data/vb/.

The most common reason to edit these files is probably to change the look and feel or the display units of some item. Here one probably only needs to edit the appropriate '.rules' file. The next most common reason is to add a new type of derived field. For this, one probably needs, at a minimum, to make changes to virtualFieldTable.txt, one or more of the \*.rules files, and browserFieldMenu.txt. For adding a new gridded data source, one needs to at least check out gridSourceTable.template. If this new gridded data source has data on previously unused levels, one would have to add these to gridPlaneTable.txt and browserPlanViewMenu.txt, and if there were new fields, one would have to add these to virtualFieldTable.txt, browserFieldMenu.txt, and the.rules files.

- 3) Rerun the `grids' task in the localization process. This will incorporate all of your changes.
- 4) Check all of the files you checked out back in.
- 5) Restart fxa to pull in all of the changes.

---

**Author: Jim Ramer**  
**Last update: 8 Jul 99**

## Sat Dirs

The following is a summary of the data that one should expect to be found in the satellite data directories. The reader should note that this arrangement is not a necessary result of any code in the satellite decoder...it is driven by the relationships set up by the satellite data keys. The source data for the satellite data keys are the files eastSatDataInfo.template and westSatDataInfo.template in \$FXA\_HOME/src/localization/nationalData. The localization scripts write the satellite data keys into the file satDataKeys.txt in the directory \$FXA\_HOME/data/localizationDataSets/\$FXA\_LOCAL\_SITE. Any changes made to these files need to be kept consistent with the depict keys, product buttons, style info, and menu entries for satellite products.

All of these directories are under \$FXA\_DATA. All of these directories have a 3 letter code for the satellite channel being used. These are as follows:

- i11 - 11 micron Window IR channel.
- i12 - 12 micron Low level water vapor channel.
- iwv - 6.7 micron upper tropospheric water vapor channel.
- i39 - 3.9 micron short wave window channel.
- vis - Visible light channel.

Some of the data in these directories is in the exact same geographic coordinates as the satellite sector we receive from nesdis; others are remapped or clipped, and this will be noted. Also, some directories contain hard links to a subset of the files in another directory; this is done to present to the user a data set with either a desired time continuity or with images that are mostly filled with data.

These directories contain raw nesdis sectors for the hemisphere:

- sat/SBN/netCDF/nhSat/nhem\_i11
- sat/SBN/netCDF/nhSat/nhem\_i12
- sat/SBN/netCDF/nhSat/nhem\_iwv
- sat/SBN/netCDF/nhSat/nhem\_vis
- sat/SBN/netCDF/nhSat/nhem\_i39

These directories contain links to sat/SBN/netCDF/nhSat/nhem\*

- sat/SBN/netCDF/nhSat/nhem\_i11/clean
- sat/SBN/netCDF/nhSat/nhem\_i12/clean
- sat/SBN/netCDF/nhSat/nhem\_iwv/clean
- sat/SBN/netCDF/nhSat/nhem\_vis/clean
- sat/SBN/netCDF/nhSat/nhem\_i39/clean

These directories contain raw North American nesdis sectors (superNat):

- sat/SBN/netCDF/superNat9/super\_i11
- sat/SBN/netCDF/superNat9/super\_i12
- sat/SBN/netCDF/superNat9/super\_iwv
- sat/SBN/netCDF/superNat9/super\_vis
- sat/SBN/netCDF/superNat9/super\_i39

These directories contain links to sat/SBN/netCDF/superNat9/super\*

```
sat/SBN/netCDF/superNat9/super_i11/clean
sat/SBN/netCDF/superNat9/super_i12/clean
sat/SBN/netCDF/superNat9/super_iwv/clean
sat/SBN/netCDF/superNat9/super_vis/clean
sat/SBN/netCDF/superNat9/super_i39/clean
```

These directories contain CONUS images, which are remapped from both the North American, and high res east/west conus sectors.

```
sat/SBN/netCDF/conusC/conus_i11/remap
sat/SBN/netCDF/conusC/conus_i12/remap
sat/SBN/netCDF/conusC/conus_iwv/remap
sat/SBN/netCDF/conusC/conus_vis/remap
sat/SBN/netCDF/conusC/conus_i39/remap
```

These directories contain links to sat/SBN/netCDF/conusC/conus\*/remap:

```
sat/SBN/netCDF/conusC/conus_i11
sat/SBN/netCDF/conusC/conus_i12
sat/SBN/netCDF/conusC/conus_iwv
sat/SBN/netCDF/conusC/conus_vis
sat/SBN/netCDF/conusC/conus_i39
```

These directories contain and high res east conus sectors. Only one version of these are kept because they are used only for clipping/remapping.

```
sat/SBN/netCDF/eastCONUS/conus_i11
sat/SBN/netCDF/eastCONUS/conus_i12
sat/SBN/netCDF/eastCONUS/conus_iwv
sat/SBN/netCDF/eastCONUS/conus_vis
sat/SBN/netCDF/eastCONUS/conus_i39
```

These directories contain and high res west conus sectors. Only one version of these are kept because they are used only for clipping/remapping.

```
sat/SBN/netCDF/westCONUS/conus_i11
sat/SBN/netCDF/westCONUS/conus_i12
sat/SBN/netCDF/westCONUS/conus_iwv
sat/SBN/netCDF/westCONUS/conus_vis
sat/SBN/netCDF/westCONUS/conus_i39
```

These directories contain and high res east conus data clipped to the regional scale.

```
sat/SBN/netCDF/eastCONUS/conus_i11/regClip
sat/SBN/netCDF/eastCONUS/conus_i12/regClip
sat/SBN/netCDF/eastCONUS/conus_iwv/regClip
sat/SBN/netCDF/eastCONUS/conus_vis/regClip
sat/SBN/netCDF/eastCONUS/conus_i39/regClip
```

These directories contain and high res west conus data clipped to the regional scale.

```
sat/SBN/netCDF/westCONUS/conus_i11/regClip
sat/SBN/netCDF/westCONUS/conus_i12/regClip
sat/SBN/netCDF/westCONUS/conus_iwv/regClip
sat/SBN/netCDF/westCONUS/conus_vis/regClip
sat/SBN/netCDF/westCONUS/conus_i39/regClip
```



# Script Override

## Introduction

Starting in the OB5 release, the means by which scripting can be overridden will become much more flexible.

If one means to override an entire subordinate script, the way this works will not change. It will remain just as has always been described in section 2 of [fileChanges](#). As described there, one can replace the entire functionality of a subordinate script only via a realization file. For example, one of the subordinate scripts is called `makeGridSourceTable.csh`. The presence of a realization file called `makeGridSourceTable.csh` will cause the functionality of that script to be totally replaced.

Here is a list of all of the subordinate scripts that are subject to this type of override, along with that task that runs the script. One should note that the subordinate script `makeScales.csh` actually does have some functionality replaced rather than augmented by `makeScales.patch`.

Script	Task
<code>makeGridSourceTable.csh</code>	<code>grids</code>
<code>makeDataSups.csh</code>	<code>dataSups</code>
<code>makeScales.csh</code>	<code>scales,clipSups</code>
<code>makeClipSups.csh</code>	<code>clipSups</code>
<code>assembleTables.csh</code>	<code>tables</code>
<code>makeTextKeys.csh</code>	<code>text</code>
<code>makeTopoFiles.csh</code>	<code>topo</code>
<code>updateGridFiles.csh</code>	<code>grids</code>
<code>updateRadarFiles.ksh</code>	<code>radar</code>
<code>makeMapFiles.csh</code>	<code>maps</code>
<code>makeWWAtables.csh</code>	<code>wwa</code>
<code>makeStationFiles.csh</code>	<code>station</code>
<code>makeDirectories.csh</code>	<code>dirs</code>
<code>createAuxFiles.csh</code>	<code>auxFiles</code>

The following subordinate scripts can have their functionality replaced, but will not source a `.patch` file.

Script	Task
<code>genRadarDataMenus.ksh</code>	<code>radar</code>
<code>genRadarDataKeys.ksh</code>	<code>radar</code>
<code>genRadarDepictKeys.ksh</code>	<code>radar</code>
<code>makeRadarSups.csh</code>	<code>radar</code>
<code>genRadarProdButtonInfo.ksh</code>	<code>radar</code>
<code>genRadarMultiLoadKeys.ksh</code>	<code>radar</code>
<code>genRadarExtensionInfo.ksh</code>	<code>radar</code>
<code>doMosaicProcessing.ksh</code>	<code>radar</code>
<code>staUtil.csh</code>	<code>station</code>
<code>wwaUtil.csh</code>	<code>wwa</code>
<code>fixGridGeo.csh</code>	<code>fixGeo</code>

## Patch scripts

Any previously written .patch scripts will behave exactly the same in OB5 as they had previously. The new feature in .patch scripts is that the localization scripting can recognize lines in .patch scripts that contain simply '#partition'. This has the effect of allowing the user to break a .patch script into parts, each part being sourced at a different point in the execution of the primary subordinate script. Previously, the user had no control over where in a subordinate script this occurred. A .patch script with no #partition lines will behave exactly as before.

The low level logic that allows this to happen is implemented in the utility program [fileMover](#) and the utility script fileGrab.csh. A .patch file with one occurrence of #partition is considered to have two partitions, first (1) and second (2). Each possible patch file has a "default partition," meaning that a file without partitions will be treated as if it has only that partition. For now, the primary partition is almost always the second (2) partition, and is almost always sourced at the end of the primary subordinate script.

Here is a table of the partitions that the localization scripting will recognize for all the currently possible .patch scripts. The table contains the script name and the partitions that will be sourced at the beginning, in the middle, and at the end of the primary subordinate script. The letter D means it is the default partition. The letter R (very rare) means that this partition will result in replacing functionality in the primary subordinate script rather than just being sourced. Sometimes the .patch scripts will be located in the file system based on the ingest site rather than the display localization ID; this is noted with an I after the primary subordinate script name.

Script		Beginning	Middle	End
-----		-----	-----	---
makeGridSourceTable.csh	I	1	2D	3
makeDataSups.csh	I	1		2D
makeScales.csh		1	2DR	3
makeClipSups.csh	I	1D		2
assembleTables.csh		1		2D
makeTextKeys.csh		1		2D
makeTopoFiles.csh		1		2D
updateGridFiles.csh	I	1	2D	3
updateRadarFiles.ksh	I	1	2D	3
makeMapFiles.csh		1		2D
makeWWAtables.csh		1		2D
makeStationFiles.csh		1		2D
makeDirectories.csh	I	1		2D
createAuxFiles.csh	I	1		2D

For the scripts that have three different places to override, this is what happens between the middle override point and the end. For makeGridSourceTable.csh, this is where the sources get turned on and off by activeGridSources.txt and inactiveGridSources.txt. For makeScales.csh, this is where the default movable points and baselines get created, and where the default Home point location is established. For updateGridFiles.csh, this is where the key and style files get generated, the families get posted to the main menu, and where all the dynamically generated

volume browser menus get created. For updateRadarFiles.ksh, this is where the radar acq params get generated, and where the national mosaic radar data keys get attached to the main radar data key file.

An important thing to keep in mind is that each possible override file is partitioned independently. It is possible to supply .patch files as realization files, site specific files, or as custom files. Before partitioning, all the scripting in these various override files would be appended and then sourced. With partitioning, these files are still all appended together, but the partitioning happens first. For example, suppose you supplied a site specific .patch script with no partitions and also the same .patch script as a custom file with a single `#partition' as the first line. This would not result in the scripting in the site specific file being treated as in partition one and the scripting in the custom file being treated as in partition two. Being unpartitioned, the scripting in the site specific file would be assigned to the default partition, usually partition two. The custom file, its first line being `#partition', would have all its scripting assigned to partition two, regardless of which partition was default or how many occurrences of `#partition' were in the site specific file.

---

**Author: Jim Ramer**  
**Last update: 29 Jun 04**

# Direct display of shape files in WFO advanced.

WFO advanced now has the ability to display shape files directly. Shape files can be used to produce three types of displays, plain vectors, labeled vectors, and labeled points. There are also now five user shape file entries on the menu, which allow the user to display a shape file directly by just placing the shapefile with the correctly named attribute(s) at the correct pathname.

The easiest way to explain how this works is to show the data key entries for the predefined user shape files and discuss the meaning of the entries. This is from the file dataInfo.manual in the directory \$FXA\_HOME/data/localization/nationalData:

```
1311 | | | | | | | |NAME      |userFile1 | |user shape file map
1
1312 | | | | | | | |NAME      |userFile2 | |user shape file map
2
1313 | | | | | | | |NAME      |userFile3 | |user shape file map
3
1314 | | | | | | | |LAT,LON,NAME |userFile4 | |user shape file map
4
1315 | | | | | | | |LAT,LON,NAME |userFile5 | |user shape file map
5
```

Each data key entry consists of one line in the file, broken into fields by vertical bars. The data key fields that are non blank refer to, respectively, the data key, the shape file attribute(s) to use, the file name of the shape file triplet (minus the .shp, .shx, and .dbf extensions), and some arbitrary identification text. For information about the meaning of other unused fields, see the header documentation in the file dataInfo.manual.

Three of these entries are set up for vector map backgrounds and two are set up for labeled points. In order to actually display a user supplied shape file, one must take the following steps:

- 1) Make sure the attributes are correct.

The eight column in a data key entry is always a list of attribute names when creating an entry for a shape file map background. If there are no attribute names present, then the software will assume that user wishes to display a plain vector map background. If one attribute name is present, then the assumption is that labeled vectors are being displayed. Three attribute names results in an attempt to display labeled points. In the case of labeled vectors, the single attribute name must refer to a text attribute and this attribute is what will be used to label the vectors; if this is not the case an error will be logged and the software will display plain vectors. If labeled points are being displayed, then the first and second attribute names must refer to floating point attributes that contain the latitude and longitude of the points, and the third must refer to a text attribute that contains the text that

will be used to label the points. In the case where one is trying to use the predefined user shape file display capability, a shape file for labeled vectors must contain a text attribute named literally `NAME'; a shape file for labeled points must contain floating point attributes named literally `LAT' and `LON', and a text attribute named literally `NAME'.

2) Put the shape file components in the proper place.

In order to utilize the user shape file display capability, the shape file components must be placed in the directory \$FXA\_HOME/data or \$FXA\_HOME/data/localizationDataSets/\$FXA\_LOCAL\_SITE. To use the first available user slot for displaying vectors from a shape file, one would end up putting the files userFile1.shp, userFile1.shx, and userFile1.dbf in this directory. To use the first available user slot for displaying labeled points from a shape file, one would end up putting the files userFile4.shp, userFile4.shx, and userFile4.dbf in this directory.

3) Make the proper menu selection.

Open the `Maps' data selection menu from the top level menu bar on the WFO advanced workstation. Find and open the submenu labeled `User Shape File'. In there there will be five product buttons labeled `Map 1' through `Map 5'. These are for loading the user shape files `userFile1' through `userFile5', respectively.

The predefined user shape file mechanism is not the only means whereby one can display shape files with the WFO advanced workstation. One can add additional shape file base map backgrounds to the workstation on a permanent basis. This can either be done as a local customization or a permanent addition to the national data set. In either case, one must make the proper data key entries, depict key entries, product button entries, and menu entries. The following table shows the files that must be edited to make each type of entry; the first file is the one used for a local customization, the second the one used for a permanent addition to the national data set. One should note that by default the environment variable FXA\_CUSTOM\_FILES should point to /data/fxa/customFiles.

Data keys:

\$FXA\_CUSTOM\_FILES/localDataKeys.txt  
\$FXA\_HOME/data/localization/nationalData/dataInfo.manual

Depict keys:

\$FXA\_CUSTOM\_FILES/localDepictKeys.txt  
\$FXA\_HOME/data/localization/nationalData/depictInfo.manual

Product buttons:

\$FXA\_CUSTOM\_FILES/localProductButtons.txt  
\$FXA\_HOME/data/localization/nationalData/productButtonInfo.txt

Menu entries:

\$FXA\_CUSTOM\_FILES/otherBackgroundMenus.txt  
\$FXA\_HOME/data/localization/nationalData/backgroundMenus.txt

Shape files:

```
$FXA_CUSTOM_FILES/*.{shp,shx,dbf}  
$FXA_HOME/data/localization/nationalData/*.{shp,shx,dbf}
```

For adding a shape file map background to the national data set, making all of the proper entries in these files should then just require a workstation restart to enable the new map background. For a local customization, one needs to run a localization with the -tables and -maps tasks as well as restarting the workstation.

For guidance in how to make these entries, one should consult the header documentation in the national data set versions of these files, and look at the entries for keys 1311 through 1315 for examples.

---

**Author: Jim Ramer**  
**Last update: 7 Dec 98**

## Static Progressive Disclosure for Point Data

When point data is displayed, we generally add additional data points/stations as we zoom up, with the attempt to add as much data as is practical without stations writing over the top of each other. The act of adding data as one zooms is referred to as progressive disclosure.

When we display point data in AWIPS, some data sets have fixed stations (like METARs) and some have locations that change with time (like ship reports). For those data that have time varying locations, we always compute the progressive disclosure on the fly. For those with fixed stations, we often will precompute the progressive disclosure, and we refer to this as static progressive disclosure. There are several reasons to precompute the progressive disclosure. First, there is a desire to have the appearance or not of a given station as one steps through frames to be the result of whether or not there is data for the given station, not the result of changes in progressive disclosure. Second, if we have a stations map background, there is a desire to have the progressive disclosure be the same in the background as in the overlaid data. Finally, static progressive disclosure allows certain important stations to always show up at low zoom levels. There are two types of static progressive disclosure files that are used by the D-2D at run time. Static plot info files are used to define the progressive disclosure for a data set, and can be used directly to plot a station map background. Location plot info files are used to plot a map background with fixed locations, such as city locations or county names. Static plot info files need to have a .spi extension, and location plot info files need to have a .lpi extension. Static plot info (.spi) files are also used to drive station selection for displaying point data in the volume browser. Files of this type can either be supplied directly or be generated by the localization task. When .spi and .lpi files are generated, they are usually generated from files with a .goodness extension. Goodness files are called this because the arbitrary user preferences in these files have acquired the name "goodness values." There are two types of goodness files: station goodness files and location goodness files. Finally, a type of file called a cities file can also be used to generate location plot info files. See [va\\_driver](#) for detailed information of the format of these file types.

The -station localization task is where all static progressive disclosure files are managed.

[mainScript.csh](#) is the script that one uses to run localizations. Most of the real work of creating static progressive disclosure files is done by the utility program [va\\_driver](#).

Here is a brief narrative of what the -station localization task does. First, .lpi files are created for labeling zones, counties, and county warning areas. Then the warnGen tables are used to generate a .goodness file for warning locations. Next, the contents of nationalData/CitiesInfo.txt and any available LocalCitiesInfo.txt file supplied as an override file are used to create a .goodness file for cities. (See the introductory section of [fileChanges](#) for a brief description of file override.) After that, .goodness files are created for the LDAD station tables and the synoptic data stations. Most of the rest of the override files are gathered next; any .lpi, .spi or .goodness files that exist in the national data set or as override files are moved into the localization data set. Each goodness file that does not have a correspondingly named .lpi or .spi file already is input to va\_driver to produce a .lpi or .spi file, depending on the format of the goodness file. The next to last thing is to make .spi files based on the ingest station tables for certain data sets. The final step is to populate static netCDF point data sets. (See the end of section 2.1 of [adaptivePlanViewPlotting](#) for more information about static netCDF point data sets.)

There are several things to note about the way file override works for these data sets. The basic CitiesInfo.txt file should never be overridden except as a realization file, and only rarely. One

should ideally override this with a LocalCitiesInfo.txt file in customFiles/. One should not replicate the whole CitiesInfo.txt file in LocalCitiesInfo.txt and start editing. One need only provide information for new cities or for those needing to be changed. An entry spelled exactly the same in both city and state will override any previous entry for that city. If a city is totally unwanted, override it with a -90.0 lat to just get it out of the way. An extra character at the end of the state field, which is ignored except for the case of determining uniqueness, can be used in the case where two different cities actually have the same name and state.

The most important thing to remember about file override for location and station goodness files is this: if there is a like named .lpi or .spi file supplied (either as a national data set or override file) the .goodness file will be ignored. By default, realization and site specific files will replace existing .goodness, .spi, or .lpi files, and those from custom files will be appended, but this can be overridden. The default replacement behavior can be changed by making the first line of the override file either #append or #replace. Just as with the cities file, if there is already an existing file of this type, one needs only to put the entries one wants to change or add in the override file, assuming the override file is appending.

In the case where one wants to have certain important stations that appear at lower zoom levels, there are a couple of strategies to try. First, one can just list the important stations in a like named .primary override file. Be advised that if one puts two stations that are very close to each other in a .primary file, one of them will still not appear until higher zoom levels. Second, one can put floating point progressive disclosure parameters in place of the goodness values for a few stations, and let va\_driver fill in the rest. It is not recommended to try to set all the progressive disclosure parameters manually. If one spends alot of time to tailor the stations just right for two or three zoom states, they will invariably produce unexpected results for other zoom states. Finally, as previously mentioned, there are several data sets for which static progressive disclosure files are automatically made from the ingest station table. When this is done, if there is already a static progressive disclosure file, then this step will add any stations in the ingest station table but not already in the existing static progressive disclosure file. This is now controlled by the contents of the file progDiscStnTables.txt, for which a default version is created in the scripting for the -station localization task. Here are the default contents of progDiscStnTables.txt:

```
metar MTR 82
maritime BUOY 84
profiler profiler 11001
raob raob 9000
modelBufr modelBufr 10001
goesBufr goesBufr 10005
poesBufr poesBufr 10006
```

Each line refers to a data set for which this processing will be done. Override files for this can be supplied and by default they append. The entries on each line are the name of the ingest station table (minus a trailing StationInfo.txt), the name of the static progressive disclosure file (minus the trailing .spi extension) and the data key for the directory where this data set is stored. It is also possible to create static progressive disclosure information from the data set itself. For now, this happens only for maritime data, and this occurs because of the existence of the file maritimeGoodnessDesign.txt, which contains instructions for creating station goodness file entries from the raw maritime data.

**Author: Jim Ramer**  
**Last update: 10 May 04**

# Style Rules

Seven style rules files control Volume Browser displays as follows:

[graphStyle.rules](#)

how to display standard graphs of a parameter versus time or versus height

[contourStyle.rules](#)

how to display contour renderings of scalar fields of gridded data

[gridImageStyle.rules](#)

how to display image renderings of scalar fields of gridded data

[iconStyle.rules](#)

how to display a scalar grid as a field of icons

[arrowStyle.rules](#)

how to display vector gridded data as either conventional or bidirectional arrows

[barbStyle.rules](#)

how to display vector gridded data as wind barbs

[streamlineStyle.rules](#)

how to display vector gridded data as streamlines

These files reside in the directory `localization/nationalData`, which can be found in `$FXA_HOME/src` in an environment with a source tree or in `$FXA_HOME/data` otherwise.

Style rules file contains two types of lines: rules and style info. Each rule should be immediately followed by style info. A rule describes a subset of all of the possible gridded data items to which to apply some style info.

## Rules

A rule line always begins with an asterisk. After that, separated by commas, appear sources, planes, and fields. To see a list of the possible source names, look at the tenth column in the file `gridSourceTable.txt`. To see a list of possible field IDs, look at the first column in the file `virtualFieldTable.txt`. To see a list of possible plane names, one really needs to run the program `testGridKeyServer`, with a single argument of ``p'`. This will direct a list of planes to standard output; the plane name is in the second column.

By default, if one never mentions any planes on a rule line, then the rule is assumed to refer to all planes. Once any planes are mentioned, then the rule refers only to those planes. The same is true for fields and sources. One can also use some special syntax with the greater than symbol for identifying a range of planes. For example, the text `850MB > 700MB` refers to all known planes with MB as their vertical coordinate and pressure values ranging from 850 to 700. One might also say `1000MB-500MB > 700MB-500MB`, but here what one gets is all of the composite (two-level) planes with MB as their vertical coordinate whose plane index ranges from that of the 1000MB-500MB plane to that of the 700MB-500MB plane. If two planes referred to in such a manner have either dissimilar plane types (standard or composite) or dissimilar vertical coordinate types, what you select is just a range of plane indices.

It is perfectly valid for two different rules to point to the same gridded data item. One might, for example, apply a rule for all instances of some field, and then make an exception for one plane or one source. When two or more different rules point to the same gridded data item, the style info for the last rule is what becomes associated with that item.

## Style Info

### **graphStyle.rules**

The style info for standard graphs has four to nine primary items separated by vertical bars. Here is a description of each item, in order. The first four fields must be present, even if blank. The rest are optional.

1. This units string replaces whatever units string is in the legend text in the depictable info table. If blank the default is left alone.
2. The data that come from the virtual data server are multiplied by this...
3. and added to by this before being displayed. This is mostly for units conversion purposes.
4. This field can have two comma-delimited parts. The first part, if non-blank, causes a log scale to be used. The second part, if non-blank, causes all items in the time series data to be summed in time, which is applicable only to time series.
5. A number which affects the minimum value of the data axis for the graph. If with a leading equals sign (=), then the minimum value must be exactly this; otherwise is must be no greater than this. If left blank, the minimum value of the data axis is totally floating.
6. A number which affects the maximum value of the data axis for the graph. If with a leading equals sign (=), then the maximum value must be exactly this; otherwise is must be no less than this. If left blank, the maximum value of the data axis is totally floating.
7. Sometimes a particular line graph may have little or no range of values. If this is the case, a minimum range of values is applied to the data axis, and this field controls that minimum range. If greater than zero, this range is an arithmetic range; if less than -1, it specifies a minimum ratio. For a logarithmic scale, this defaults to -10, otherwise to 10. This is ignored if inconsistent with entries made in the previous two fields (minimum and maximum data axis values).
8. When a logarithmic scale is made for data that goes to or through zero, there is a break between the log part of the graph and the linear part immediately around zero. This entry, when greater than zero, controls this value, which otherwise the code will determine based on the characteristics of the data being displayed. For a linear scale, this is a value which the data axis need not contain, but if extended indefinitely a major division will fall exactly there. In either case, this defaults to zero if not defined. This is ignored if inconsistent with entries for minimum and maximum data axis values.
9. A list of up to 3 comma-delimited values, which apply only to time series, not parameter vs height graphs. Whenever one of these values appears within the range of values on the data axis, a dotted horizontal line will appear at that value. The existence of these values has no direct effect on how the data axis is chosen, so if one needs to be sure they always appear on the graph, then the minimum and maximum data axis values need to be set appropriately.

## contourStyle.rules

The style info for contour data has eleven primary items separated by vertical bars. Here is a description of each item, in order.

1. This units string replaces whatever units string is in the legend text in the depictable info table. If blank the default is left alone.
2. The data that come from the virtual data server are multiplied by this...
3. and added to by this before being contoured. This is mostly for units conversion purposes.
4. This is the approximate number of labels that will be placed on a contour that is as long as the display is wide.
5. This is the format control string for labels on the contours. Format control strings look a lot like Fortran format descriptors, owing to the fact that the contouring routine is in Fortran. See [the description](#) at the end of this document for more about format control strings. This will default to something reasonable if left blank.
6. Format control string for labeling maxes and mins. Just an X results in using the same thing as for labels. Blank means no labels for maxes and mins.
7. What to use to mark mins and maxes. First character is mins, second character is maxes. Use a dot for a place holder to indicate no mark for either mins or maxes. An @ will suppress either maxes or mins from being labeled even if there is a non-blank entry in item 6.
8. This eight-digit hexadecimal number controls the line style. The last four digits represents the bit pattern of the line style. A leading 8 means use solid for positive contours and the bit pattern for negative contours.
9. This item contains up to six sub-items separated by commas. It is allowable to leave this blank, or provide fewer than six items; the missing items (or items with all white space) will revert to their default values. Here is a description of each sub-item, in order.
  - i. If non-blank, this allows one to supply a list of positive contour values, and all of the corresponding negative values will be contoured as well.
  - ii. The display width in km to which the default contouring interval applies, defaults to 5000km.
  - iii. Exponent that determines the response to zooming of the contour interval. A value of 1 means a two to one zoom results in halving the contour interval; a value of 0.5 means a four to one zoom is required to halve the contour interval.
  - iv. Smoothing distance. If greater than zero, a smoother will be applied to take out perturbations smaller than this distance in km. If less than zero, the smoother will operate on perturbations smaller than that many grid points. 0 means no smoothing.
  - v. If non-blank, this makes this display sampleable. A digit specifies the precision of the sampling: number of digits past the decimal for typical magnitude numbers, literal precision for large magnitude numbers.
  - vi. By default, a sample string is identified with the virtual field table ID. Here one can provide an alternate ID for the sample string. A single period means format the sample string with no ID. This sub-field is not meaningful if the previous sub-field is blank.

10. Number of contour values. If negative, then this is the approximate number of contours desired, and the user is expected to supply a minimum contour increment in the next item. If 0, the user is expected to supply a contour increment in the next item. If 1000, then the user is expected to supply three values in the next item: an increment and a range of values to contour. If positive, then the user is expected to supply a list of that many contour values to use. If a negative value is used here, a single contour increment will be used for the entire loop of some particular product.
11. Contour increment or list of values to contour, space-delimited.

It should be noted that the contour increment (11) and the labeling frequency (4) are what should nominally appear at one to one zoom on the CONUS scale with density and magnification both set to one. Different display scales, zooms, densities and/or magnifications will cause these to be adjusted.

### **gridImageStyle.rules**

The style info for image renderings of gridded data has ten primary items separated by vertical bars. Here is a description of each item, in order.

1. This units string replaces whatever units string is in the legend text in the depictable info table. If blank the default is left alone.
2. The data that come from the virtual data server are multiplied by this...
3. and added to by this get the display units. Color bar is labeled in these units.
4. Value to which to map the minimum pixel count. Two values separated by a greater than sign will cause this to vary with the vertical coordinate value. Presence of the item `log' in the rule (delimited by commas like everything else) will cause this to vary with the natural log of the vertical coordinate value.
5. Value to which to map the maximum pixel count. Two values separated by a greater than sign will cause this to vary with the vertical coordinate value. Presence of the item `log' in the rule (delimited by commas like everything else) will cause this to vary with the natural log of the vertical coordinate value.
6. This item contains up to two sub-items separated by commas. If the first is non-blank, mapping from data to counts is logarithmic. If this feature is used, then the values to which to map the minimum and maximum pixel counts must both be positive and the minimum pixel count value must be smaller than the maximum pixel count value. The second item is a smoothing distance. If greater than zero, a smoother will be applied to take out perturbations smaller than this distance in km. If less than zero, the smoother will operate on perturbations smaller than that many grid points. 0 (the default) means no smoothing.
7. This item contains up to four sub-items separated by commas. It is allowable to leave this blank, or provide fewer than four items; the missing items (or items with all white space) will revert to their default values. Here is a description of each sub-item, in order.
  - i. If non-blank, this allows one to supply a list of positive label values for the color bar, and all of the corresponding negative values will be labeled as well. This mode also allows one to use a log scale for negative values.

- ii. If non-blank, values that map to a color index less than 1 will be black; by default, they take on a color index of 1.
  - iii. If non-blank, values that map to a color index greater than 254 will be black; by default, they take on a color index of 254.
  - iv. If non-blank, data will be displayed as a pixelated (non-interpolated) image.
8. Default color table index. \$FXA\_HOME/data/colorMaps.mark is a commented CDL file which can be used to tell which color table indices go with which color table.
9. Labeling control flag:

<0 The range of data values in the grid is determined and then the data are scaled to cover that range minus 20 percent. This is to allow for the fact that once a scaling is determined for one frame in a loop for a particular product, that scaling will be used for each frame.

=0 The next item is the labeling increment.

>0 The next item is a list of values to label on the color bar.

>20 This is an ImageStyle key that defines the color bar labeling, in image counts. This is usually used where the data being displayed are represented by an enumeration, like the hydrometeor class. Item 10 needs to be empty in this case.

10. Labeling increment or list of values to label on the color bar, space-delimited.

## iconStyle.rules

The style info for rendering of gridded data as a field of icons has six primary items separated by vertical bars. Here is a description of each item, in order.

1. This units string replaces whatever units string is in the legend text in the depictable info table. If blank the default is left alone.
2. The data that come from the virtual data server are multiplied by this...
3. and added to by this before being rendered. This is mostly for units conversion purposes.
4. Number of different characters to use as icons. An optional second comma-delimited parameter is the [character set](#) to use. 0 is regular ASCII, 1 is large ASCII, 2 is weather symbols, 3 is special symbols, and 4 is large special symbols.
5. List of values, space-delimited and in ascending order, for which icons will be plotted.
6. List of characters, as a contiguous string, that will be used as icons for the values listed in item 5. Alternatively, may be a list of spaced-delimited ASCII character codes to use.

## arrowStyle.rules

The style info for rendering of vector gridded data as a field of arrows has three primary items separated by vertical bars. There are also two optional additional fields that may be supplied. Here is a description of each item, in order.

1. This units string replaces whatever units string is in the legend text in the depictable info table. If blank the default is left alone.

2. The data that come from the virtual data server are multiplied by this to get the display units. If this is a negative number then logarithmic scaling will be used for the arrow lengths.
3. This is the magnitude value, in display units, that is scaled to the default length, which is 25 pixels times the character magnification. Two values separated by a greater than sign will cause this to vary with the vertical coordinate value. Presence of the item `log' in the rule (delimited by commas like everything else) will cause this to vary with the natural log of the vertical coordinate value.
4. This is the magnitude value, in display units, that is the smallest value for which to show an arrow. Defaults to zero.
5. This is the magnitude value, in display units, that is the largest value for which to show an arrow. Defaults to an arbitrarily large number.

### **barbStyle.rules**

The style info for rendering of vector gridded data as a field of wind barbs has four or five primary items separated by vertical bars. Here is a description of each item, in order.

1. This units string replaces whatever units string is in the legend text in the depictable info table. If blank the default is left alone.
2. The data that come from the virtual data server are multiplied by this to get the display units for placing ticks and flags on the barbs.
3. This is the magnitude value, in display units, that is the smallest value for which to show a barb. Defaults to zero.
4. This is the magnitude value, in display units, that is the largest value for which to show a barb. Defaults to an arbitrarily large number.
5. If non-blank, this makes this display sampleable. A digit specifies the precision of the magnitude sampling: number of digits past the decimal for typical magnitude numbers, literal precision for large magnitude numbers. By default, a sample string is identified with the virtual field table ID. If an additional comma delimited sub-field is provided here, that is an alternate ID for the sample string. A single period means format the sample string with no ID.

### **streamlineStyle.rules**

The style info for rendering of vector gridded data as streamlines has four primary items separated by vertical bars. Here is a description of each item, in order.

1. If greater than one, no two streamlines will approach any closer than this number of cells.. If less than one, a streamline will terminate if it runs through 1/minspc consecutive already-occupied cells.
2. No streamline will be started any closer than this number of cells to an existing streamline.
3. This is the magnitude value, in virtual data units, that is the smallest value for which to show streamlines. Defaults to zero.

4. This is the magnitude value, in virtual data units, that is the largest value for which to show streamlines. Defaults to an arbitrarily large number.

## Processing Style Rules

After making changes in a rules file, one needs to run the ``-grids'` task in the localization to implement the changes. In the normal diagnostic feedback from the ``-grids'` task is a message ``running processStyleInfo'`. Any syntactic problems with changes to the rules files will show up as error messages here.

If there are syntactic problems with a rules file it may be useful to run the command

```
textBufferTest rulesFile > tempFile
```

This will yield a file where continuations are resolved and comments have been eliminated; thus it can be directly compared to the line numbers that are referred to in diagnostics from `processStyleInfo`. The program `textBufferTest` builds in `D-2D/src/util` and should be in `$FXA_HOME/bin` at a remote installation.

## Format Control Strings

An example of a label format control string is `"2:f5.2;"`. The `f5.2` is the Fortran format to be used for generating labels. The `2` means that two characters are removed from the front of the string generated by the format before leading spaces are stripped. The `;` on the end means that trailing zeroes after the decimal point are NOT removed, which of course is meaningful only for a floating format. If no `:` or `;` is present, the string is assumed to contain only a Fortran format. The routine will provide defaults for any of these specifications that are omitted. Currently, internal work arrays limit labels to 6 characters in length.

Some examples:

Value	FCS	Result
1004.2	"i4"	"1004"
1004.2	"2:i4"	"04"
3.50	"f6.2"	"3.5"
3.50	"f6.2;"	"3.50"
3.50	"1:f6.2;"	"3.50"
3.50	"1:f4.2"	".5"

---

**Author: Jim Ramer**  
**Last update: 16 Oct 07**

# Text Templates

In WFO Advanced, the warnGen application is used to generate the text of warnings, watches, or advisories (WWAs) whose issuance requires direct geographic interaction with a weather display. This includes virtually all of the short term severe weather warnings. WarnGen uses template files to control exactly how the text of each WWA is created. Template files allow one to change the characteristics of a particular WWA, or add a new one, without having to change the code.

There are four main concepts to understand within template files: [paragraphs](#), [substitutions](#), [variables](#), and [bullets](#).

## Paragraphs

The text of a template is very free format. In general, consecutive spaces are changed to one space before processing, spaces preceding a period are removed, and all consecutive lines of text without an intervening blank line are considered to be in a paragraph. Also, individual lines and arguments in substitutions have their leading and trailing spaces stripped before processing. A place holder character (~) and a paragraph break character (&) are available to override this default behavior. Later in this document is a table of all special characters.

## Substitutions

A substitution is a signal to the software to build some text based on the geographic, temporal, or other characteristics of the WWA in question. The general format of a substitution is as follows:

```
< substitution_type | qualifier_type = qualifier_value | ... >
```

The substitution type, qualifier type, and qualifier value are in general just text. However, certain qualifier types for certain substitution types do result in the qualifier value being interpreted as a number. Not all qualifier types require that a qualifier value be present. Normally, leading and trailing spaces are stripped off of the qualifier value. However, if a `==` (double equals sign) is used between a qualifier type and value, then a leading and trailing space are added to the qualifier value.

If a line or series of lines contains nothing but a substitution that results in no text being generated, it is as if those lines never appeared in the template. Thus, a null substitution will not create a paragraph break in this case. If it is desired that text from two substitutions be directly adjacent with no intervening spaces, then the trailing delimiter from the first needs to be directly adjacent to and on the same line as the leading delimiter from the next. A line continuation (backslash at the end of a line, see special characters table) can accomplish this as well.

To date, by convention, substitution types are all caps and qualifier types are all lower case with underscores. The text that results from a substitution can be completely within a paragraph, be a single paragraph in itself, or span several paragraphs. Place holder and/or paragraph break

characters in the text from a substitution are fully interpreted in organizing paragraphs. Later in this document is a table of currently available substitutions. Two very important types of substitutions rely on geographic entity lookup tables (GELTs) to produce their text. Some additional information about GELTs is available in [newGELTmaker](#). Additionally, the reader is directed to the entry for the [VAR](#) substitution. The purpose of this substitution is to assign values to template [variables](#) (see next section); the functionality of this substitution type has recently been greatly enhanced.

## Variables

While it is not possible to nest substitutions, it is possible to direct the text of a substitution into a variable, and then that variable can be referred to within the value of a `lead' or `trail' qualifier in another [substitution](#). The text of any substitution can be directed to a variable by placing a qualifier of the type `var' in the substitution. The value of the `var' qualifier is the name of the variable to which the text of the substitution is assigned. When directing the text of a substitution to a variable, that text will not appear in the output unless that variable is later referenced. Variables can also be referenced in any plain text outside of substitutions. Variable names should be all alphabetic or numeric characters, with no escape sequences or spaces (underscores are OK). A variable name is referred to with a leading `\$\$' (double dollar sign) and a trailing `!' (exclamation point). There are several [predefined template variables](#) that are set up by either the localization or the warnGen application that are helpful in composing templates. When generating the final output text, all interactions between variables and any substitution not in an inactive [bullet](#) (see next section) are resolved in order to determine the value of the variable each time it is referred to. During the initial parse that determines the state of the bullets in the GUI, whether substitutions are within bullets is ignored. Thus, where template variables are used to determine the initial state of the warnGen GUI, it is recommended to rely on predefined template variables or variables whose value is not determined within bullets. The new enhanced functionality of the [VAR](#) substitution now allows the template writer much more control over the value of variables outside bullets.

## Bullets

A bullet is a piece of text that can appear or not appear in the output text based on a software switch under control of the user. The basic format of a bullet is as follows:

```
{<tag>=trigger= ^[txt.op.txt] title text | bullet text }
```

In this idealized format, all the non-alphabetic characters are literal. Everything in a bullet is optional, except for the leading and trailing curly, and the vertical bar separating the title text from the bullet text.

The essential characteristics of a bullet are what text it generates as output (bullet text), whether that text will actually be included in the final output text (activation state), what text will be presented in the warnGen GUI relating to the bullet (title text), whether the bullet will be presented in the GUI (show state), and whether the user will be able to change the initial

activation state from the GUI (lock state). By default, bullets are initially inactive, do not show up on the GUI, and are not locked.

The bullet text is what actually appears in the output text should the final state of the bullet be active. The bullet text can contain any plain text and any complete substitutions, but cannot contain other bullets. Just as with a substitution, the text from a bullet can be completely within a paragraph, be a single paragraph in itself, or span several paragraphs. Substitutions can appear within the text of a bullet, but bullets cannot be imbedded within substitutions. The bullet text must be between the first vertical bar and the trailing curly. If one puts variable-defining substitutions into the text of a bullet, then the behavior of those substitutions depends on whether one is initially parsing the template or generating text. In the initial parse, these substitutions will assign values to their variables as if they were not inside a bullet. At text generation time, whether that substitution is used to assign a value to the variable is controlled by whether that bullet is activated.

The title text can appear anywhere before the initial vertical bar, and after any optional tag and/or trigger designations. Variables that appear in this text will be translated, but only at GUI initialization time. The caret symbol (^) can appear in this same area, and its presence means the initial activation state of the bullet is to be included in the output text; otherwise, the initial activation state will be inactive. For those bullets that appear on the GUI, the user can toggle their activation state unless their lock state is to be locked.

The logical operator designator ( [txt.op.txt] ) can also appear in the area after the tag and/or trigger designations and before the initial vertical bar. These logical operators make it possible for bullets to be hidden from the user interface and to have their states controlled by the contents of template variables, which includes predefined template variables and environment variables. A logical operator designator should have one of the following formats:

```
[show xxx.oo.yyy], [on xxx.oo.yyy], [toggle xxx.oo.yyy], [lock xxx.oo.yyy], or  
[xxx.oo.yyy].
```

Any text not matching one of these formats will be interpreted as the plain text part of the title. The xxx and yyy are any arbitrary text that may or may not contain template variables (normally at least one will contain a variable). The `oo' is one of the following logical operators: **eq**, **ne**, **gt**, **lt**, **ge**, **le**, **in** (left string is contained in right string), or **ni** (left string is not contained in right string). The operators **gt**, **lt**, **ge**, and **le** are purely string comparisons. Only one logical operator designator is allowed in a bullet title. The one with the lead keyword `show' means that if the test is true, then present that bullet in the user interface. The one with the lead keyword `on' means if the test is true, then that bullet should be on (active) by default. The `toggle' test will present that bullet in the user interface *and* toggle the default activation state as determined by the leading caret sign (^) if true. The `lock' test will result in the activation state of the bullet being unchangeable in the GUI if the test is true. The remaining type of test with no keyword will result in the bullet being withheld from the GUI (show state is false), and whether the bullet is active (has its text included in the final output) will depend on the truth of the test. A new feature recently implemented is that for any logical operator other than **in** and **ni** one can have the the

arguments be interpreted numerically if the operator is upper case; that is **EQ**, **NE**, **GT**, **LT**, **GE**, or **LE**.

The trigger designator allows one to specify that the initial activation state of the bullet can be controlled by the contents of a previous text product. The leading equals sign (=) of the trigger designator must be immediately after the leading curly ({} of the bullet if there is no tag designator, or immediately after the closing angle bracket (>) of the tag designator if there is one. If the trigger designator is only a single equals sign, then this will invoke a default algorithm for choosing the exact text that triggers the initial state of the bullet to be active. Otherwise, the text that falls between the two equals signs is the trigger text. The trigger text can be divided into multiple strings, each of which must be present in the text to activate the bullet, using a comma (,) as a delimiter. Finally, one can place a leading minus sign (-) on a trigger string, which means that this text must not be present to activate the bullet. For bullets that show up on the user interface, this controls only the initial activation state of the bullet; the user can change it if the bullet is not locked.

Associating tags with bullets is a recently implemented feature. The tag is used to give the bullet an identifier, and this identifier can be used in conjunction with predefined patterns of template variables to control various characteristics of the bullet. This is especially useful if one has already entered a logical operator designator for the bullet and there are still other characteristics that one wants to independently control. Also, because it is possible to give the same tag to several bullets, one can use this feature to control the characteristics of several different bullets all together. The leading < of the tag designator must be immediately after the leading curly of the bullet, and there must be no non-alphabetic characters within the tag designator. Suppose that the text for the tag designator was <mytag>. Then the value of the template variable named mytag\_\_on could be used to determine the initial activation state of that bullet, assuming it was not an empty string. If the parse time translation of mytag\_\_on were FALSE or OFF, then the bullet would be initialized as inactive; otherwise, any translation other than an empty string would initialize it as active. Other available tag suffixes are \_\_show, \_\_lock, and \_\_trigger. The reader should note that these tag suffixes have two underscores in them, and that a parse time translation for a tag variable of an empty string will never have any effect. In this example, if the value of mytag\_\_trigger were NOTRIGGER, controlling the activation state by previous text would be disabled; otherwise, the parse time translation of mytag\_\_trigger would become the trigger text. The value for mytag\_\_show would be a simple binary control for that state – parse time translations of FALSE or HIDE for the show state variable would remove the bullet from the GUI; otherwise, it will be present. For the variable mytag\_\_lock, parse time translations of FALSE, FREE, or UNLOCK would let the user control its activation state from the GUI; any other value except RADIO would lock the initial activation state. The value RADIO would cause all bullets that have the same tag to have their activation states coordinated with radio button functionality. Where a logical test and a tag variable both affect the same characteristic, the tag variable takes precedence unless the logical test was to lock the bullet and the tag variable was to give it radio button functionality.

## **Title Lines**

For products that are not on the very top level warnGen menu (in the Other: selector), the title that appears in that menu is controlled by the first line in the template file, which has the following format:

```
// "sort text | title text"
```

Any text before the optional vertical bar does not actually appear on the warnGen menu, but just controls the sorting of the products in the menu. If a template has no title, then no entry corresponding to it will appear in the Other: product selector.

## Special characters for template files

\

Backslash at the end of a line represents a line continuation to the module that actually reads the text file into memory. Line continuations are meaningless in the context of the template and can cause odd paragraphing behavior, so this use is not recommended. In all other cases, a backslash escapes the immediately following character. This means that the following character will appear in the text without the backslash, but will not be interpreted as a special character.

//

Double slash is a comment marker in the module that actually reads the text file into memory. From there to the end of the line is a comment. Comments are OK, but they should be used with care inside a substitution or bullet.

\$\$

Double dollar sign is the signal that what follows is a variable name, and that it should be replaced with the value of that variable in the output text. The variable name is up to the next escape or exclamation point.

!

Variable name terminator. In the case where a `!' is used to terminate a variable name, the exclamation point does not appear in the output text.

#

When appearing at the beginning of a line, the pound sign can be part of a C/C++ style include statement, which is interpreted by the module that actually reads the text file into memory.

<>

Substitution delimiters. Text in angle brackets does not appear in the output directly. Substitution text is a description of some text which can be built based on the geographic, temporal, or other characteristics of the warning, watch, or advisory in question. Also, immediately after the leading curly for a bullet, these serve as bullet tag delimiters.

{ }

Bullet delimiters. Text in curly braces can appear or not appear in the output text based on a software switch under control of the user, or based on logical operators embedded in the bullet title.

|

Field separator. Separates title from the text in a bullet, individual qualifiers from each other in a substitution.

=

Separates the type of a qualifier from its value. Also, when immediately following the lead curly or optional tag designer in a bullet, allows the state of the bullet to be controlled by the text of a previous product.

^

Marks a bullet as being included by default.

,

Comma delimits individual trigger strings when the lead curly or optional tag designer of a bullet is immediately followed by an equals sign, and then another equals sign to designate trigger text.

~

Indent/place holder. When internal to a paragraph, will cause a space to be placed where otherwise the automatic paragraph formatting might cause a space to be removed. At the beginning of a paragraph, causes all text in that paragraph to be indented one space for each ~ that appears. An escaped space will behave just like a place holder, but not like an indent marker.

%

Reverse indent marker. When at the beginning of a paragraph causes all text in that paragraph except for the first line to be indented one space for each % that appears. Reverse indent markers can appear immediately after standard indent markers (~ characters).

&

Paragraph break. Causes a new paragraph to start without an intervening blank line. Two consecutive paragraph breaks will force a blank line to appear.

[]

Used as translation delimiters within [translation control strings](#), and as delimiters for logical expressions in bullet titles.

.

Delimits logical operators within logical expressions that may appear in bullet titles.

`

The backquote is an invisible character, which will result in no output text but will still be treated as a non-null piece of text for the purposes of formatting the output from a GELT.

## Available substitutions

Each type is followed by a sub table describing the applicable qualifiers. Qualifiers require no value unless qualifier values are mentioned. We break these into three broad categories: those that have no specific meaning in the template code itself but have meaning because of the way WarnGen is implemented, those that are meaningful in the template code in the absence of WarnGen, and those that are obsolete.

## Substitutions dependent on the WarnGen implementation

TWO\_TIMES

This substitution generates no text. When present, the 'Change...' dialog in the warnGen interface becomes active. The beginning and ending times in this dialog are accessible in the template using the START and EXPIRE substitutions.

#### DURATIONS

This is a special substitution that generates no text. Each qualifier is a possible duration for the WWA, in number of minutes or in hh:mm format. The qualifier with the value 'default' is the default duration. (This substitution type inverts the usual syntax for qualifier types and values.)

#### BEGIN\_MOTIONLESS

This substitution generates no text. The presence of this substitution will tell warnGen to allow feature tracking, but to initialize it with no motion. The user will still need to drag the point to the desired area, but until one moves the tracking icon on a different frame, the motion will remain exactly zero. This substitution is meaningless if a MOVEMENT substitution is not present.

#### CORRECT

This substitution generates no text, and is meaningful only in original issuance (not followup) templates. When present, warnGen will attempt to decode earlier versions of the text product being issued here for the purpose of issuing corrections.

#### REISSUE

This substitution generates no text, and is meaningful only in original issuance (not followup) templates. When present, warnGen will attempt to decode earlier versions of the text product being issued here for the purpose of issuing additional products if the weather event continues beyond the original expiration. This also activates the ability to do corrections.

#### COMBINED

This substitution generates no text. When present, it is a signal to warnGen that this template is meant to produce text that refers to two separate weather phenomena, and can be used only to correct or reissue products that also refer to two separate weather phenomena.

#### XXXMATCH

This substitution generates no text. Rarely, there will be a situation where multiple text products that refer to the same VTEC event are stored in the text database with different AFOS XXXs. When this occurs, it is usually the result of the site having recently undergone a transition in either its WMO ID or the primary XXX it uses for issuing products. When present, this substitution tells warnGen to allow this template only to correct and/or follow up products with the same XXX as it will generate.

#### SEGMENTED

This substitution generates no text. When present, warnGen will assume the text product being generated will be of segmented format, meaning the UGC and VTEC occur after the MND header. Currently, warnGen can generate products of this type only with a single segment, except for the SLS, which has predefined geographic segmentation.

#### ETN\_STATIC

This substitution generates no text. Normally warnGen will attempt to update the event tracking number in the VTEC when products are issued that are not corrections or followups. When this substitution is present, this updating will not occur – whatever ETN the template generates will be used.

## DEPICT\_KEYS

This is a special substitution that generates no text. Each qualifier is a map background key that should be loaded in the warnGen program when this template is being used. Multiple occurrences of this substitution cause all the unique mentioned keys to accumulate as map background keys, unless a key has a qualifier of `remove', in which case the key will be removed as a map background key.

## AUX\_INFO

This is a special substitution that generates no text. Each qualifier is a key, and each value is some text that can be passed back to the client based on that key. This substitution encompasses a great deal of functionality; see the [complete treatment of AUX\\_INFO](#) below. At a minimum, to function correctly in warnGen, the *issue\_prod* key must have a value corresponding to the CCCNNNXXX of the product being created. One or more space-delimited CCCNNNXXXs occurring with the *follow\_prods* key specifies that this template will be used to issue followup products for those text product types.

## TextTemplate essential substitutions

The following six types are time generating substitutions. All have the same list of possible qualifiers.

### ISSUE

Causes text to be generated describing the issue time of the text product being generated.

### START

Causes text to be generated describing the start time of the text product being generated.

### EXPIRE

Causes text to be generated describing the expiration time of the text product being generated.

### EVENT

Causes text to be generated describing the time of occurrence of the weather event for which the text product is being generated.

### PURGE

Causes text to be generated describing the purge time of the text product being generated.

### NOW

Causes text to be generated describing the current time.

— Qualifiers for time generating substitutions. —

#### clock

Output in clock format, e.g. 905 PM MDT. Default. Will add a day of week if code detects that day may be ambiguous.

#### header

Output in product header format, e.g. 255 PM MDT WED JUL 12 1995.

#### ydmthmz

Output in full VTEC format, yymoddThhmmZ, where yy is year, mo is month, dd is day, hh is hour, and mm is minute.

#### dthmz

Output in abbreviated VTEC format, ddThhmmZ, where dd is day, hh is hour, and mm is minute.

ddhhmm

Output in like format, where dd is day, hh is hour, and mm is minute.

plain

Output including a plain language description of the time of day, e.g. 200 PM MDT WEDNESDAY AFTERNOON.

abstime

Output a decimal integer representation of a time that can be used with the relational comparisons that appear in bullet title lines. If there is an argument, then it is assumed to be a decimal UNIX time to use as the time value, in which case this controls the value of the time to output rather than the format.

local

Output in local time. Default.

gmt

Output in Greenwich Mean Time (UTC).

interval

Value is number of minutes to which to round time. By default times are not rounded.

round

Same as interval, but causes the resulting time to be used to alter the internally held value for the time being output.

delta

Value is number of minutes by which to change the time specified by the substitution type. Default is zero. A floating point number between -2 and 2 means units are fraction of the duration of the watch or warning.

update

Whatever time is computed for this as a result of interval, round, or delta, force that time into the type indicated by the value.

last\_table

Output this time text only if the last area table used produced some output.

no\_text

Will not generate any text.

value

If the substitution successfully produces text and this qualifier is present, the text produced will be replaced with this text.

lead

Value is some arbitrary text that will precede the time description.

trail

Value is some arbitrary text that will follow the time description.

var

Value is the name of a variable that has assigned to it the text produced here.

TIME\_ZONE

This substitution generates no text; its purpose is to control how and whether time zone information is put into formatted time strings.

no

If present, do not output time zone information.

yes

If present, do output time zone information. Default.

change

If present, output time zone information when the time zone changes. Will always output a time zone the first time after this is invoked.

force

Output time zone information using a specific UNIX time zone environment variable, the text of which is in the value.

## VAR

This substitution is made available for the purpose of allowing the user to direct text into a variable. This substitution has recently had its functionality enhanced to include the ability to perform logical tests to determine the resulting value of the variable.

test

Contains a logical test formatted just as in the title line for a bullet, but without the [] delimiters. All consecutive test qualifiers immediately before a value qualifier must be logically true before that value qualifier is used.

value

The text that becomes the translation assigned to the variable. If there are multiple value qualifiers, then the first one that has all its immediately preceding test qualifiers true will be used. In the absence of test qualifiers, the first one is used.

lead

Text prepended to the translation assigned to the variable.

trail

Text appended to the translation assigned to the variable.

var

Value is the name of a variable that has assigned to it the translation produced here. If none of the value qualifiers has all its preceding test qualifiers logically true, then no text will be generated and no value will be assigned to the variable. In the absence of any test and value qualifiers, the text assigned to the variable will be a catenation of the lead and trail qualifier values, which gives backward compatibility to previous implementations of the VAR substitution.

## DISTANCE\_UNITS

This substitution generates no text; its purpose is to control the units with which distances are reported.

units

Units string to attach to distances, defaults to `MILES'.

multiplier

Number by which to multiply raw distance values to get the desired unit. Raw distances are in km, so default multiplier is 0.6211 (1/1.61).

## MOVEMENT

This substitution generates text that describes the movement of the weather event for which the WWA is being generated. If the movement is marked as undefined, no text will be generated. The presence of a MOVEMENT substitution is what triggers the appearance on the screen of the tracking object.

units

Units string to attach to speed, defaults to `MPH'.

multiplier

Number by which to multiply raw speed from tracking calculation to get desired unit. Raw speed is in km/s, so default multiplier is 2236 (3600/1.61).

interval

Value is number of speed units to which speed is rounded. By default, speeds are not rounded except to an integer.

stationary

If this is text, then it is the text used to describe a stationary weather event. If a number, then it is the speed in output units below which a weather event is considered stationary. Defaults to `STATIONARY` and 2.5.

move\_lead

Value is some arbitrary text that will precede the speed and direction description for a weather event that is not stationary. Defaults to `MOVING~`.

move\_trail

Value is some arbitrary text that will follow the speed and direction description for a weather event that is not stationary. Defaults to an empty string.

value

If the substitution successfully produces text and this qualifier is present, the text produced will be replaced with this text.

lead

Value is some arbitrary text that will precede the entire movement description.

trail

Value is some arbitrary text that will follow the entire movement description.

var

Value is the name of a variable that has assigned to it the text produced here.

**POLYGON**

Puts encoded latitude and longitude coordinates into the product that can be used for plotting warnings.

**STORM**

Puts encoded storm motion (degrees and speed in knots) and latitude and longitude coordinates of the storm location at issue time into the product.

**COORDS**

Just like putting both the POLYGON and STORM substitutions into the product.

**COLUMNS**

This is a special substitution that generates no text. It controls column layouts.

lead

Text that occurs before the first column. Defaults to `~~` (two place holders).

separator

Text that occurs between columns. Defaults to `~` (one place holder).

trail

Text that occurs after the last column. Defaults to an empty string.

**AREA**

This substitution causes text to be generated describing the area of the WWA. This substitution makes use of a GELT.

file

Value is the name of a GELT file, minus the file suffix. One can use UNIX environment variables within the file name. There can be several `file' qualifiers in one `AREA' substitution. With one exception, all qualifiers up to the next `file' qualifier modify that `file' qualifier. The default behavior is to process each `file' qualifier in order until one is found that actually generates some text, then return that text for the substitution. When two consecutive file qualifiers refer to the same file, the values of other qualifiers will be preserved. They will often revert to defaults when a new file is introduced. If this is not true, then this will be noted and that qualifier will be referred to as persistent.

accumulate

If this qualifier is present, then all file qualifiers will be processed and the list of items used will be the sum of the items produced from all of the tables. This qualifier is persistent.

area

This qualifier defines the area of interest for which this GELT file will try to provide a description. If the value is `WWA' then the base polygon of the WWA is used to define the area of interest. If the value is some other GELT file, then whatever area is currently held within that GELT is used as the area of interest. This is useful for imposing consistency between different GELTs that might otherwise react differently to filtering because they have different types of geographic entities. If this qualifier is not present, whatever area is currently held within that GELT is used, unchanged. The first time a GELT file is referred to within a template, this qualifier needs to be present. This qualifier is persistent even to file changes.

format

The purpose of this qualifier is to control how lists of individual items from a GELT are put together. The value can be `list' (default), `ugc', `count', `none', `simple', `xxx\_columns', or `blank'. `list' will cause each item returned to be put in a list separated by ellipses. `ugc' will cause the items to be formatted as if they were a list of UGC codes. `xxx\_columns' will cause the items to be arranged in columns, where the actual text of the value is `one\_column', `two\_columns' up through `seven\_columns'. `count' just returns an ASCII string representing the number of items in the list. `simple' means just concatenate the text together. `none' means no text is generated for this file qualifier. `blank' means no text is generated for the list, but lead and trail qualifiers will still be used. This qualifier is persistent.

multiple

If the value is `yes', then the substitution will produce text only if more than one point describes the weather event. If the value is `no', then the substitution will produce text only if a single point describes the weather event. Any other value invokes the default behavior, which is to allow any number of points in the weather event. If the weather event does not exist, then this qualifier has no effect. This qualifier is persistent.

min\_count

Minimum number of unique items that must result from the GELT query after translation in order to allow the current qualifier to generate text. Defaults to one. When in accumulate mode and the value is negative, accumulation will stop as soon as that many (absolute value) are present. This qualifier is persistent.

max\_count

If positive, and more than this many items are returned from the GELT query, no text will result. If negative, will truncate the list to that many (absolute value). When in accumulate mode and the value is positive, will dispose of only the text from the specific `file' qualifier that caused the count to exceed the threshold, not all text. Defaults to a very large number. This qualifier is persistent.

output\_field

Each geographic entity in a GELT has some descriptive text associated with it, which is broken into one or more fields delimited by vertical bars. The value of the `output\_field' qualifier is the index of the field which is the text returned for each geographic entity, one based. Zero (the default) means return all text regardless of field delimiters.

item\_format

This value is a [translation control string](#) which controls how each item from the GELT is reformatted. The default is to do no reformatting. This is really a more powerful version of the `output\_field' qualifier.

sort\_by

This value is a [translation control string](#) which controls how the individual items from the GELT are sorted before being used. If blank, then no sorting occurs, which is the default behavior. This qualifier is persistent.

stratify\_by

This value is a [translation control string](#) which controls one manner by which the individual items from the GELT are grouped before being used. Items for which the result of the translation is the same are considered to be in the same group. For each group, it is as if a separate substitution entry were present, with the formatting and application of lead/trail qualifiers occurring independently. If blank, then no stratification occurs, which is the default behavior. This qualifier is persistent.

group\_by

This value is a [translation control string](#) which controls yet another manner by which the individual items from the GELT are grouped before being used. This manner of grouping responds to the sort\_by qualifier. Consecutive items that have the same result of the group\_by translation are in the same group. These groups do not respond as if each were from a separate substitution entry. An item's position in a group can be acted on by the item\_format translation control string. If blank, all items are in the same group, which is the default behavior. This qualifier is persistent.

in\_group

If positive, and more than this many items in any group result from the GELT query, no text will be returned. If negative, will truncate each group to that many items (absolute value). When in accumulate mode and the value is positive, will dispose of only the text from the specific `file' qualifier that caused the maximum group size to exceed the threshold, not all text. Defaults to a very large number. This qualifier is persistent.

max\_groups

If positive, and more than this many groups result from the GELT query, no text will be returned. If negative, will truncate the list to that many groups (absolute value). When in accumulate mode and the value is positive, will dispose of only the text from the specific `file' qualifier that caused the group count to exceed the threshold, not all text. Defaults to a very large number. This qualifier is persistent.

unique\_by

The result of the [translation control string](#) in a unique\_by qualifier is the way in which items can be marked as non-unique and be removed. If blank, then it will not be used, which is the default behavior. This qualifier is persistent.

delta

Invokes a feature which causes a time, distance, and bearing to be assigned to each geographic entity, based on when the weather event will be closest to that entity. If a value is present, entities having a time associated with them within that many minutes of the start time of the warning will not be used. The time, distance, and bearing can be used by a [translation control string](#). This qualifier is persistent.

interval

If present, times assigned to geographic entities will be rounded to this many minutes.

This qualifier is persistent.

max\_dist

A point more than this far (in km) from the location of the weather phenomenon will not be referenced. This also causes a time, distance, and bearing to be assigned to each geographic entity, based on when the weather event will be closest to that entity. This qualifier is persistent.

adapt\_dist

Allows an item to be removed based on a special maximum distance only if specified text either occurs or does not occur in the text for the item. A number specifies the pertinent distance in km. A string that begins with a plus or minus will test only against the rest of the string. A string that begins with a minus will test true if it is not in the raw text, otherwise strings test true if they are in the raw text. Multiple test strings can be supplied with multiple adapt\_dist qualifiers. To be false, only one text-not-there test must fail; to be true, only one text-there test must succeed.

group\_alone

Allows one to specify that an item must be in a group all by itself in order to be included, based on whether specified text either occurs or does not occur in the text for the item. A string that begins with a plus or minus will test only against the rest of the string. A string that begins with a minus will test true if it is not in the raw text, otherwise strings test true if they are in the raw text. Multiple test strings can be supplied with multiple group\_alone qualifiers. To be false, only one text-not-there test must fail; to be true, only one text-there test must succeed.

proximal

Value is an additional phrase prepended to the description of a single point if it happens to be directly over some geographic entity. If a number, then this is how close (in km) a weather event must be to a location to be considered "OVER" it. The default values are "OVER~" and 3 km. This qualifier is persistent.

portions

If present, activates a feature which will provide a plain language description of which portions of a geographic entity fall within the area of interest. A value, if present, represents the minimum size in square km that an entity must be in order to be described in this fashion, the default being zero.

central

If present, activates a feature which will allow the use of the keyword "central" when describing a portion of an area.

extreme

If present, activates a feature which will make use of the keyword "extreme" when describing the situation where only a very small portion of some geographic entity falls within the area of interest.

min\_fraction

Value is the minimum fraction of a geographic entity which must fall within the area of interest for that geographic entity to be included. Default value is zero, so if this qualifier is not included, then the feature is in effect turned off.

min\_area

Value is the minimum size in square km of a geographic entity which must fall within the area of interest for that geographic entity to be included. Default value is zero, so if this qualifier is not included, then the feature is in effect turned off.

test\_both

When present, a portion of a geographic entity must pass both tests to be included in the area of interest. By default, it must pass only one test or the other.

value

If the substitution successfully produces text and this qualifier is present, the text produced will be replaced with this text. This qualifier is persistent.

lead

Value is a [translation control string](#) which will provide text that will precede the text provided by the GELT. First item from the GELT is input to the translation control string to produce the result. This qualifier is persistent.

trail

Value is a [translation control string](#) which will provide text that will follow the text provided by the GELT. Last item from the GELT is input to the translation control string to produce the result. This qualifier is persistent.

include\_field

It is possible to include only those geographic entities for which a certain text fragment occurs in a certain field; the value of this qualifier is the index of that field. This index affects the next `include\_text' qualifier that occurs. The default is 0, which causes all fields to be checked.

include\_text

Text to look for in the field pointed to by the last `include\_field' qualifier. The occurrence of the `include\_text' qualifier is what actually activates this feature. Multiple occurrences of the `include\_text' qualifier for a single `file' qualifier will result in multiple checks for text that must be present in an entity. The default behavior is for no include checks to occur. The effect of this qualifier never carries over between `file' qualifiers.

exclude\_field

It is possible to exclude those geographic entities for which a certain text fragment occurs in a certain field; the value of this qualifier is the index of that field. This index affects the next `exclude\_text' qualifier that occurs. The default is 0, which causes all fields to be checked.

exclude\_text

Text to look for in the field pointed to by the last `exclude\_field' qualifier. The occurrence of the `exclude\_text' qualifier is what actually activates this feature. Multiple occurrences of the `exclude\_text' qualifier for a single `file' qualifier will result in

multiple checks for text that must not be present in an entity. The default behavior is for no exclude checks to occur. The effect of this qualifier never carries over between `file' qualifiers.

no\_same

If present, will not generate any text if the last time a GELT-based substitution was used the identical text was generated.

cross

This allows an additional table to be used to add information to individual item descriptions. In the cross-reference table, the description is of the centroid of the entity found in the main table in the `file' qualifier. See the description of [translation control strings](#) for information about how to use the information from a cross-reference table in an item description.

used

This qualifier allows one to exclude geographic entities that were previously used to generate text in some other instance of an AREA or WX substitution. There are three main values for this qualifier: `clear', which means empty out the list of previously used entities; `accumulate', which means add entities from this substitution to the list; and `avoid', which means do not generate text for those entities in the list. There are also two hybrid values for this qualifier: `begin', which means clear then accumulate; and `implement' which means avoid then clear.

var

Value is the name of a variable that has assigned to it the text produced here.

area\_handling

Value is how the input area is used to update the current active area held by the GELT. Defaults to `initialize', which means set the active area to be the same as the input area. `add' means add the input area to the current active area. `remove' means remove the input area from the current active area. `restrict' means keep in the current active area only those areas both already in the current active area and in the input area. `toggle' means change the active state of all points in the input area.

sequence

When present, this qualifier activates a feature whereby potentially redundant text can be removed from the items returned. If there is grouping as defined with the `group_by` qualifier, then redundant text can be removed from all but one item within the group. Any text before a leading `FROM' before any occurrence of `TO' will be removed from all but the first of the group. Any text after any occurrence of `TO' and starting with the occurrence of one of the delimiting strings supplied as arguments to any number of sequence qualifiers will be removed from all but the last of the group. Delimiting strings supplied as arguments beginning with an underscore allow redundant text to be removed from the beginning of an item when it occurs both at the beginning and end of the item. Text after the underscore in this case is strings that must occur in the redundant text for it to be removed.

WX

This substitution causes text to be generated describing the location of the weather event for which the WWA is being generated. This substitution makes use of a GELT.  
file

Value is the name of a GELT file, minus the file suffix. One can use UNIX environment variables within the file name. There can be several `file' qualifiers in one `WX' substitution. All qualifiers up to the next `file' qualifier modify that `file' qualifier. The code will process each `file' qualifier in order until the required minimum number of weather points has been identified. When two consecutive file qualifiers refer to the same file, the values of other qualifiers will be preserved. They will often revert to defaults when a new file is introduced. If this is not true, then this will be noted and that qualifier will be referred to as persistent.

area

This qualifier defines the area of interest for which this GELT file will try to provide a description. If the value is `WWA', then the base polygon of the WWA is used to define the area of interest. If the value is some other GELT file, then whatever area is currently held within that GELT is used as the area of interest. This is useful for imposing consistency between different GELTs that might otherwise react differently to filtering because they have different types of geographic entities. If this qualifier is not present, whatever area is currently held within that GELT is used, unchanged. The first time a GELT file is referred to within a template, this qualifier needs to be present.

format

Value is either `list', `line', or `none', of which `line' is the default. `list' will cause a description of the locations of weather events to be presented as an ellipsis-delimited list. `line' will cause a list of weather locations to be described as a line spanning these locations. `none' will result in no text being generated. This qualifier is persistent.

multiple

If the value is `yes', then the substitution will produce text only if more than one point is being used to describe the weather event. If the value is `no', then the substitution will produce text only if a single point is being used to describe the weather event. Any other value invokes the default behavior, which is to allow any number of points. This qualifier is persistent.

output\_field

Each geographic entity in a GELT has some descriptive text associated with it, which is broken into one or more fields delimited by vertical bars. The value of the `output\_field' qualifier is the index of the field which is the text returned for each geographic entity, one based. Zero means returns all text regardless of field delimiters. This qualifier is persistent.

item\_format

This value is a [translation control string](#) which controls how each item from the GELT is reformatted. The default is to do no reformatting. This is really a more powerful version of the `output\_field' qualifier. This qualifier is persistent.

portions

If present, activates a feature which will provide a plain language description of which portions of a geographic entity fall within the area of interest. A value, if present, represents the minimum size in square km that an entity must be in order to be described in this fashion, the default being zero.

extreme

If present, activates a feature which will make use of the keyword "extreme" when describing the situation where a point falls very near the boundary of some geographic entity.

#### min\_fraction

Value is the minimum fraction of a geographic entity which must fall within the area of interest for that geographic entity to be included. Default value is zero, so if this qualifier is not included, then the feature is in effect turned off.

#### min\_area

Value is the minimum size in square km of a geographic entity which must fall within the area of interest for that geographic entity to be included. Default value is zero, so if this qualifier is not included, then the feature is in effect turned off.

#### test\_both

When present, a portion of a geographic entity must pass both tests to be included in the area of interest. By default, it must pass only one test or the other.

#### value

If the substitution successfully produces text and this qualifier is present, the text produced will be replaced with this text. This qualifier is persistent.

#### lead

Value is a [translation control string](#) which will provide text that will precede the text provided by the GELT. First item from the GELT is input to the translation control string to produce the result. This qualifier is persistent.

#### trail

Value is a [translation control string](#) which will provide text that will follow the text provided by the GELT. Last item from the GELT is input to the translation control string to produce the result. This qualifier is persistent.

#### include\_field

It is possible to include only those geographic entities for which a certain text fragment occurs in a certain field; the value of this qualifier is the index of that field. This index affects the next ``include_text'` qualifier that occurs. The default is 0, which causes all fields to be checked.

#### include\_text

Text to look for in the field pointed to by the last ``include_field'` qualifier. The occurrence of the ``include_text'` qualifier is what actually activates this feature. Multiple occurrences of the ``include_text'` qualifier for a single ``file'` qualifier will result in multiple checks for text that must be present in an entity. The default behavior is for no include checks to occur. The effect of this qualifier never carries over between ``file'` qualifiers.

#### exclude\_field

It is possible to exclude those geographic entities for which a certain text fragment occurs in a certain field; the value of this qualifier is the index of that field. This index affects the next ``exclude_text'` qualifier that occurs. The default is 0, which causes all fields to be checked.

#### exclude\_text

Text to look for in the field pointed to by the last ``exclude_field'` qualifier. The occurrence of the ``exclude_text'` qualifier is what actually activates this feature. Multiple occurrences of the ``exclude_text'` qualifier for a single ``file'` qualifier will result in multiple checks for text that must not be present in an entity. The default behavior is for no exclude checks to occur. The effect of this qualifier never carries over between ``file'` qualifiers.

#### filter

If present, activates behavior where no item in a GELT can be referred to unless it is identified as being at least partially within the area currently held by the GELT.

no\_same

If present, will not generate any text if the last time a GELT-based substitution was used the identical location description was generated.

proximal

Value is an additional phrase prepended to the description of a single point if it happens to be directly over some geographic entity. If a number, then this is how close (in km) a weather event must be to a location to be considered "OVER" it. The default values are "OVER~" and 3 km. This qualifier is persistent.

max\_dist

A point more than this far from the location of the weather phenomenon will not be referenced. This qualifier is persistent.

adapt\_dist

Allows an item to be removed based on a special maximum distance only if specified text either occurs or does not occur in the text for the item. A number specifies the pertinent distance in km. A string that begins with a plus or minus will test only against the rest of the string. A string that begins with a minus will test true if it is not in the raw text, otherwise strings test true if they are in the raw text. Multiple test strings can be supplied with multiple adapt\_dist qualifiers. To be false, only one text-not-there test must fail; to be true, only one text-there test must succeed.

group\_alone

Allows one to specify that an item must be in a group all by itself in order to be included, based on whether specified text either occurs or does not occur in the text for the item. A string that begins with a plus or minus will test only against the rest of the string. A string that begins with a minus will test true if it is not in the raw text, otherwise strings test true if they are in the raw text. Multiple test strings can be supplied with multiple group\_alone qualifiers. To be false, only one text-not-there test must fail; to be true, only one text-there test must succeed.

interval

If present, a reported time for the weather event is rounded to this many minutes. This qualifier is persistent.

delta

Normally, a `WX' substitution will describe the location of the weather event as it was identified on the last frame with the storm marker. If the `delta' qualifier is present, then this substitution will describe the projected location of the weather event that many minutes in the future. This substitution will not generate text if the resulting time is not within the valid period of the WWA.

used

This qualifier allows one to exclude geographic entities that were previously used to generate text in some other instance of an AREA or WX substitution. There are three main values for this qualifier: `clear', which means empty out the list of previously used entities; `accumulate', which means add entities from this substitution to the list; and `avoid', which means do not generate text for those entities in the list. There are also two hybrid values for this qualifier: `begin', which means clear then accumulate; and `implement' which means avoid then clear.

var

Value is the name of a variable that has assigned to it the text produced here.

sequence

When present, this qualifier activates a feature whereby potentially redundant text can be removed from the items returned. If there is grouping as defined with the `group_by` qualifier, then redundant text can be removed from all but one item within the group. Any text before a leading ``FROM'` before any occurrence of ``TO'` will be removed from all but the first of the group. Any text after any occurrence of ``TO'` and starting with the occurrence of one of the delimiting strings supplied as arguments to any number of sequence qualifiers will be removed from all but the last of the group. Delimiting strings supplied as arguments beginning with an underscore allow redundant text to be removed from the beginning of an item when it occurs both at the beginning and end of the item. Text after the underscore in this case is strings that must occur in the redundant text for it to be removed.

## Obsolete substitutions

LOCK

This substitution is now obsolete. Please see the last paragraph in the [AUX\\_INFO section](#) for information about how to lock editing of the polygon if this is not the case by default.

UNLOCK

This substitution is now obsolete. Please see the last paragraph in the [AUX\\_INFO section](#) for information about how to allow editing of the polygon if this is not the case by default.

FREE\_TIMES

This substitution is now obsolete. Please see the last paragraph in the [AUX\\_INFO section](#) for information about how to allow editing of the times if this is not the case by default.

UGC\_FREE

This substitution is now obsolete. Please see the last paragraph in the [AUX\\_INFO section](#) for information about how to allow editing of the UGCs if this is not the case by default.

## Translation Control Strings

This section describes how translation control strings work. As mentioned before, each piece of text found in the \*.id file of a GELT is broken up into fields by vertical bars. A translation control string allows the user to intermingle literal text, untranslated text straight from one of the fields of an item of GELT text, or an item of GELT text that is translated somehow.

Here is an example of a translation control string:

```
ABC [1]DEF[2,tuv] HIJ [xyz]
```

The result of this translation control string will be the literal text "ABC ", followed by the contents of field one, followed by the literal text "DEF", followed by the contents of field 2 acted on by the translation type ``tuv'`, followed by the literal text " HIJ ", followed by the result of

translation type `xyz' acting on the whole GELT item. A field index of 99 is treated the same as literal text.

There are several numbers that can be added to the field index that will allow additional flexibility in formatting. Adding 50 to the field index will cause the formatting code to try to get a field from cross-reference text. This will work only if a `cross' qualifier is present with the table from which one is generating text. 50 means all text from the cross-reference table, 51 means field one from the cross-reference table, etc.

The other numbers that can be added to field indices control whether text is generated based on an item's position in a group. Adding 1000 means show this text only if the item's current group has exactly one item. Adding 2000 means show this text only if the item's current group has more than one item. Adding 3000 means show this text only if there is just one weather point. Adding 6000 means show this text only if there is more than one weather point.

Group positions are referred to as *front*, *back*, and *mid*, referring to the first, last, and anything not first or last, respectively. Additionally, *start* and *end* refer to the first and last in the entire lists of items. These are the numbers that can be added to a field index to refer to the various positions in a group:

```
100 - start or front
200 - start
300 - front
400 - mid
500 - back
600 - end
700 - back or end
```

If one of these numbers is added to the field index, then the item must be in that position in the group for that text to be shown. If the field index is made negative, then the item must not be in that position in the group for that text to be shown.

Here is a list of the recognized translation types. For some of them, the text of the GELT item is not actually used.

state

This translation attempts to convert an upper case postal abbreviation into a state name. The data for this translation are in the file `nationalData/state.abrev`.

area

This translation attempts to convert a lower case abbreviation of an area of a state into plain language, such as `ne' to `NORTHEAST'. The data for this translation are in the file `nationalData/areas.abrev`.

county\_type

This translation attempts to identify the state in question and provide the proper nomenclature for counties or county equivalents (e.g. PARISH). The data for this translation are in the file `nationalData/county_type.abrev`.

counties\_type

Same as `county\_type` translation except that it will return plural if the number of items returned from the GELT query is more than one.

area\_state

This translation combines the results of the `state` and `area` translation and attempts to provide text such as `NORTHEAST TEXAS`.

county\_area\_state

This translation combines the results of the `state`, `area`, and `county\_type` translation and attempts to provide text such as `BACA COUNTY IN SOUTHEAST COLORADO`.

clock

Provides a clock format description of the local time. Uses the beginning of the WWA for the `AREA` substitution, uses the time of the weather event for a `WX` substitution. This translation type does not use the text of the GELT item.

plain

Just like a `clock` translation type except it provides a plain language description of the date and time.

header

Just like a `clock` translation type except it provides a header format description of the date and time.

ddhhmm

Provides a ddhhmm format description of the UTC time. Uses the beginning of the warning for the `AREA` substitution, uses the time of the weather event for a `WX` substitution. This translation type does not use the text of the GELT item.

count

Returns an ASCII representation of the count of the number of items returned from a GELT query.

gcnt

Returns an ASCII representation of the count of the number of items in a group.

index

Returns an ASCII representation of the GELT table index of an item. Useful only in a `sort\_by`, `group\_by`, or `unique\_by` qualifier.

-index

Returns an ASCII representation of the negative of a GELT table index of an item. Useful only in a `sort\_by` qualifier.

gidx

Returns an ASCII representation of the position in its group for an item.

itime

Returns an ASCII representation of the UNIX time associated with an item. Useful only in a `sort\_by`, `group\_by`, or `unique\_by` qualifier.

-itime

Returns an ASCII representation of the negative of the UNIX time associated with an item. Useful only in a `sort\_by` qualifier.

lat

Returns an ASCII representation of the latitude associated with an item. Useful only in a `sort\_by` qualifier.

-lat

Returns an ASCII representation of minus the latitude associated with an item. Useful only in a `sort\_by` qualifier.

lon Returns an ASCII representation of the longitude associated with an item. Useful only in a `sort\_by` qualifier.

-lon Returns an ASCII representation of minus the longitude associated with an item. Useful only in a `sort\_by` qualifier.

size Returns an ASCII representation of the size in hectares of an entity (to the resolution of the GELT grid). Useful only in a `sort\_by` qualifier.

-size Returns an ASCII representation of the negative of the size in hectares of an entity. Useful only in a `sort\_by` qualifier.

table Returns an ASCII representation of the order of the `file` qualifier in use within the substitution. Useful only if in accumulate mode and for a `sort\_by`, `group\_by`, or `unique\_by` qualifier.

-table Returns an ASCII representation of the negative of the order of the `file` qualifier in use within the substitution. Useful only if in accumulate mode and for a `sort\_by` qualifier.

dist Returns an ASCII representation of the distance from the weather event in tenths of km. Useful only for a `sort\_by`, `group\_by`, or `unique\_by` qualifier.

-dist Returns an ASCII representation of the distance from the weather event in tenths of km. Useful only for a `sort\_by` qualifier.

azran Adds description of distance and bearing in miles and degrees meteorological from an entity to the weather event. For zero distance will add proximity descriptor.

azrn0 Adds description of distance and bearing only if distance is greater than zero.

azrn1 Adds description of distance and bearing. Uses proximity descriptor if distance is zero; strips portion of area descriptor otherwise.

azrn2 Adds description of distance and bearing only if distance is greater than zero; strips portion of area descriptor.

azrn3 Describes distance and bearing; uses proximity descriptor if distance is zero.

azrn4 Describes distance and bearing only if distance is greater than zero.

county\_count Returns an ASCII representation of the count of the number of items returned from a GELT query, followed by the same output one would get from the `counties\_type` translation.

alpha

Removes all non-alphabetic characters.

S

Blank if the item count is less than two, otherwise returns "S".

s

Blank if the item count is less than two, otherwise returns "s".

## Predefined template variables

Normally, when one refers to AFOS text product IDs (CCCNXX) in templates, one will use template variables that look like `$$NNNid!`, where NNN is the AFOS product type. The localization creates entries that define the correct values for these template variables in the include file `$(CURRENT_CWA)-offIncl.txt`, which lives in `localizationDataSets/LLL/`. `CURRENT_CWA` points to the CWA for which one is currently issuing products, and `LLL` is the localization in use; all templates should include this file. For example, for the BOU county warning area, the translation of `$$SVSid!` will be `DENSVSBOU`. The localization will create definitions for `$$NNNid!` for each NNN that appears in the file `nationalData/wgn_wwa_NNN.txt`.

The localization will also create definitions for the variables `$$cccValue!` and `$$xxxValue!`, which are the primary CCC and XXX with which the warnGen and WWA applications issue products. There will sometimes also be definitions for `$$ccc2Value!` and `$$xxx2Value!`, which are an optional secondary CCC and XXX. Certain legacy templates will use `$$cccValue!NNN$$xxxValue!` where AFOS text product IDs are referred to, but this syntax can cause problems in the case where there is not complete uniformity in the CCCs and XXXs being used for TextTemplate-generated products at the site.

Additionally, the variable `$$wmoValue!` points to the primary WMO ID being used to issue products for the site. The values of `$$wmoValue!`, `$$cccValue!`, `$$xxxValue!`, `$$ccc2Value!`, and `$$xxx2Value!` are controlled, respectively, by the WMO, CCC, XXX, CCC2, and XXX2 directives. See [directives.html](#) for more information on directives.

The variables that have been discussed so far are defined within the context of localization and have their definition pulled into the templates through include files. The remainder of these variables are ones that are set as environment variables within the context of the warnGen application.

Among these are predefined template variables that point to things that have to do with the VTEC coding. The action code for the primary VTEC line is found in the template variable `$$ACT_VAL!`. Currently supported action codes for warnGen are NEW (new event), COR (correction), EXT (extension in time), CON (continuance), CAN (cancellation), and EXP (expiration). These action codes are used extensively in the default templates to make decisions about which text to show based on the action being taken. At text generation time, `$$ACT_VAL!` is always defined, whether the product being created is based on a previously issued product or not; at parse time it will often be an empty string. By defined, we mean it refers to something other than an empty string.

There are also several predefined variables that are defined only when the product being created is based on a previously issued product. The variable `$$PREV_TEXT!` will translate to a single string containing all of the text from the previously issued product, up to 5000 characters. `$$ACT_VAL2!`, is the action code for the second event of a followup product. `$$VTEC_EVENT!` and `$$VTEC_EVENT2!` are VTEC event strings for the first and second events for a followup product. An example of a VTEC event string would be `KTOP.TO.W` for a Topeka tornado warning. `$$PPS_VAL!` is the primary VTEC phenomenon and significance (e.g. `TO.W`) of the product being followed up and `$$NNN_VAL!` is the text product type (e.g. `TOR`) of that product. `$$HYDRO_VAL!` will contain the text of the hydro VTEC line, `$$CAUSE_CODE!` will contain the two letter immediate cause code and `$$TEXT_CAUSE!` will contain the plain text of the immediate cause. Of these, only `$$VTEC_EVENT!`, `$$PPS_VAL!`, and `$$NNN_VAL!` are guaranteed to be defined every time a previous product is involved. `$$ACT_VAL2!` and `$$VTEC_EVENT2!` are defined only when the previous product refers to a second weather event, `$$HYDRO_VAL!` is defined only when a hydro VTEC line exists, `$$TEXT_CAUSE!` exists only when plain language is found in the product that refers to an immediate cause, and `CAUSE_CODE` is defined only when either the text cause can be decoded or there in an immediate cause coded in the hydro VTEC line.

Finally, the variable `$$MND_VAL!` contains the MND header qualifier. `$$MND_VAL!` should always be at the end of the product type line in the MND header. This is set with a plain language indication of when a product is text, experimental, or corrected.

## **AUX\_INFO functionality**

As was previously mentioned, the values for the keys found in the `AUX_INFO` substitution drive a very diverse set of functionality, and as such we have this separate section to describe how the `warnGen` application responds to these various keys. By convention, there is only one `AUX_INFO` substitution present in the main files of all the default templates. However, there can be multiple instances of this substitution, and in theory multiple instances of any key as well. When this occurs, the last value encountered for that key is what is used.

The keys `geo_descriptor`, `wwa_type`, `wx_hazard`, `specific_hazard`, and `ugc_codes` at one time were how the `warnGen` application told the MDL Watch, Warning, and Advisory (WWA) application about the products it is generating. Their presence is not harmful, but these no longer have any effect because the WWA application has been decommissioned.

The single most important key is `issue_prod`; its value is the `CCCNXX` of the text product that this template is designed to create. Optionally, one can add a second space-delimited item here that is the VTEC event (e.g. `KMSP.FL.Y`) for the product being issued.

Occasionally, text products other than the product being issued or the product(s) being followed up will need to be parsed to determine everything about a given VTEC event. As an argument to the key `parse_prods`, one can provide a space-delimited list of `CCCNXX`s for additional product types that should be parsed to determine everything about the given VTEC event.

The presence of the *follow\_prods* key is what designates a template as being primarily for creating followup products instead of original-issuance products. The value for this is a space-delimited list of either text product IDs or VTEC events for which one wishes to use this template to create followup products.

Warngen now supports two new types of followup actions – corrections for followup products and extensions in time. Neither of these is activated by default in the software; one must use the *follow\_actions* key in the template to activate them. The value of this key is a list of additional non-default followup actions to activate for the template – COR for corrections and EXT for extensions in time. An action must actually be supported before putting it in this list; it is expected that future releases will support additional actions.

The *text\_cause* key allows one to specify that a previous product used with the template must either have or not have a plain text immediate cause present. If the value is YES, the previous product must have a plain text immediate cause; if the value is NO, then the previous product must not have a plain text immediate cause.

The *cause\_codes* key allows one to specify lists of immediate cause codes that either must or must not be in the previous product for it to be used with the template. A value of NONE means that the previous product must not have a cause code to be used; a value of ANY means that the previous product must have some cause code to be used. Otherwise, the value is a list of space-delimited immediate cause codes, one of which must be the code in the previous product in order for it to be used with the template. A leading space-delimited NOT means that none of the codes in the list can be in the previous product for it to be used.

There is now the ability to exert fine scale control over what characteristics of the product one is issuing should remain unchangeable from the time of the last Restart or selection from the followup action list. The characteristics that one can control in this manner are the times, the polygon, the text bullets, whether one can add UGCs, and whether one can remove UGCs. Each of these characteristics is represented by a suffix in the key: *\_time*, *\_polygon*, *\_bullets*, *\_ugcadd*, and *\_ugcdel*, respectively. The prefix in the key is the action being undertaken. The recognized actions are NEW, COR, CON, CAN, EXP, and EXT. If one is correcting anything but a new product, the prefix is compound. Thus, for example, if the entry CORCON\_times=free appeared in an AUX\_INFO substitution, this would allow editing the times for correcting a continuance. The entry EXT\_polygon=lock would lock the polygon for a new extension in time, and COR\_ugcadd=lock would prevent one from adding UGCs to a correction to a new product. All of these states have well-defined default values so usually templates will not need these entries, but if these entries are made they will override whatever the defaults are.

---

**Author: Jim Ramer**  
**Last update: 04 Dec 2006**

# WarnGen Backup

Historically, there have been two primary types of backup in WarnGen, Full backup and Partial backup. Partial backup is no longer considered consistent with the current concept of operations, and so the selectors that invoke it have been removed from the WarnGen GUI as of OB8.2. Furthermore, the selector on the WarnGen GUI that invokes Full backup is now entitled simply 'Backup'. Many sites may retain some partial backup data structures, which ideally should be removed. For most sites, the `usa_cwa_total.*` shape file set should be deleted from `nationalData/`; **failing to do so will reduce the geographic precision of the WarnGen tables.** A few sites may continue to use the `usa_cwa_total.*` shape file set to delineate downstream areas outside their CWA for which they have Dam Break warning responsibility for dams inside their CWA. In this case, the partial backup *data structures* are used to support this; one DOES NOT NEED the partial backup option on the WarnGen menu to issue these types of warnings.

For those sites that need to use the `usa_cwa_total.*` shape file set to delineate downstream areas outside their CWA for which they have Dam Break warning responsibility, there are some things to be aware of. This capability is invoked by two customizations. First, the downstream counties need to be placed in the `usa_cwa_total.*` shape file set for the site. Other than counties in your CWA, ONLY those downstream counties should be included in the area defined for your site in that shape file. It may be tempting to just rely on an older version of `usa_cwa_total.*` shape file set that has entire neighboring sites included, but this will result in the aforementioned problems with geographic precision. Second, for any template that one needs the expanded effective CWA, the line containing:

```
#include "wwa_county_ugc.template"
```

needs to be changed to say:

```
#include "wwa_county_ugc_stretch.template"
```

While this functionality was originally designed specifically for the dam break template (by default, `wwa_dam_break`), this change can be made in any template. If a site will retain follow-up responsibility for those downstream counties, then this change should also be made in the non-convective FFS template (by default, `wwa_fflood_sta`) as well. Once configured in this manner, one does not need to use any of the backup functionality on the warnGen GUI to issue dam break warnings for these counties outside your CWA; just draw the warning to include the counties.

For short-fused warnings, Full backup is the preferred method of backup for a short unscheduled takeover of backup responsibilities. For an extended scheduled backup, it may be preferable to restart a whole D-2D for the backup site.

When one selects a Backup site on the WarnGen menu, the WarnGen program will exit and restart running under the localization ID of the full backup site. If one is running WarnGen for a backup site, the Backup selector will be colored yellow.

The availability of sites for full backup is controlled by two things. First, the desired full backup site must be listed in the SBID directive in the `wwaConfig.txt` file (see [directives](#) for more on this). Second, one must have run the localization for the full backup site.

When one runs a localization for a full backup site, it is important to get the arguments right. Suppose that LLL is the primary localization ID for your site, and that BBB is the localization ID for some site for which you want to do full backup. Here are three examples of how you might run a localization for this purpose:

```
./mainScript.csh -WWA BBB
./mainScript.csh BBB
./mainScript.csh -wwa BBB
```

The first command is the most common way to run the localization for the backup site the very first time you set it up for full backup. Note that the task identifier is -WWA in all caps, which will run enough localization tasks for a site to support full backup, as opposed to -wwa, which will just try to regenerate the WarnGen tables. The second command would be used in the case where you also desire to run a D-2D as the backup site. The third command is what you would use if the site was already established as a working full backup site but you need to update templates or perhaps regenerate the geographic tables because of new shape files.

Normally at an installed site, one will always run localizations with only one localization ID on the command line. However, for development, testing, or certain other unusual situations, one might run a localization with two localization IDs on the command line. If this is the case, the command needed to run a localization that can support the full backup function must be this:

```
./mainScript.csh -WWA BBB LLL
```

and NOT this:

```
./mainScript.csh -WWA BBB BBB
```

There are some gotchas to watch out for in configuring a system for Full service backup. First, you have to have the correct format for the SBID directive in the LLL-wwaConfig.txt file. Suppose that the entire set of sites for which you wanted to enable full backup is BBX, BBY, and BBZ. Here is the proper format for the SBID directive:

```
@@@SBID "BBX", "BBY", "BBZ"
```

The following formats are NOT correct:

```
@@@SBID "BBX, BBY, BBZ"
```

```
@@@SBID BBX, BBY, BBZ
```

```
@@@SBID "BBX", "BBY", "BBZ"
```

```
@@@SBID "LLL", "BBX", "BBY", "BBZ"
```

The last incorrect example will work but will result in multiple entries for your default site appearing in the `Backup' selector.

Another gotcha is using bad customization practices for the wwaConfig.template file. First of all, unless you are changing the set of templates that you want to appear on the very top level of the WarnGen GUI, there is no reason to customize this file. By default, any file in customFiles/ or localization/LLL/ with a correct pathname and properly formatted title line will automatically get posted to the Other: list on the WarnGen GUI. See [TextTemplate](#) for more on the format of templates. Bad customization practices in this file can break this feature. If it is deemed absolutely essential to override wwaConfig.template, you should change only the lines directly applicable to posting templates to the very top level of the GUI. Especially, do not change or remove any lines that look like these:

```
int = numMajorProds, index, count
SeqOfTextString = prodTypeNames, prodTypeFileNames, cwaIds

"XXX000", \
"YYY000", \

warngen.cwaIds: ###CWAS
```

To know whether your site has an override file for wwaConfig.template, look for the following pathnames:

```
/data/fxa/customFiles/wwaConfig.template
/data/fxa/customFiles/LLL-wwaConfig.template
/awips/fxa/data/localization/LLL/LLL-wwaConfig.template
```

It is an especially bad idea to modify nationalData/wwaConfig.template in place.

---

**Author: Jim Ramer**  
**Last update: 12 Feb 08**

## binary cartographic data files &

### extended binary cartographic data files

This program is a stand-alone program for performing operations on WFO Advanced .bcd (binary cartographic data files) or .bcx files (extended binary cartographic data files), which are the file formats that the workstation actually reads to draw map backgrounds. A .bcd file contains vector data in lat/lon coordinates. A .bcx is essentially a .bcd file with a text string associated with each record. Both input and output are assumed to be the same file type.

The program allows the user 6 different operations which can be performed on either .bcd or .bcx files except where noted. These are summarized here and explained in detail below:

- remove identical vectors
- clip based on a depictor file
- collapse short vectors
- clip an ASCII file based on a depictor file
- clip an ASCII file based on a depictor file with lat/lon units conversion
- shorten records to a standard length
- convert an ASCII file to a bcd file

For the mode that removes identical vectors, the usage is as follows:

```
bcdProc r {x} input_file output_file
```

r: the letter `r'

x (optional): if the letter `x' is present, assume .bcx file.

input\_file: pathname of file to read.

output\_file: pathname of file to write.

For the mode that clips based on a depictor file, the usage is as follows:

```
bcdProc c {x} input_file depictor output_file
```

c: the letter `c'

x (optional): if the letter `x' is present, assume .bcx file.

input\_file: pathname of file to read.

depictor: pathname of depictor file (.sup file) to clip with.

output\_file: pathname of file to write.

For the mode that collapses short vectors, the usage is as follows:

```
bcdProc t {x} input_file dist output_file
```

t: the letter `t'

x (optional): if the letter `x' is present, assume .bcx file.  
input\_file: pathname of file to read.  
dist: vectors shorter than this distance in km will be collapsed.  
output\_file: pathname of file to write.

This mode reads lines from an ASCII file, finds first two consecutive space-delimited words that can be a lat/lon, and removes the record if that lat/lon is not within the area of the depictor file. To be a lat/lon, a word must contain a decimal point and represent a floating point number within the allowable range of values for lats and lons in degrees. The usage is as follows:

```
bcdProc a input_file depictor output_file
```

a: the letter `a'  
input\_file: pathname of file to read.  
depictor: pathname of depictor file (.sup file) to clip with.  
output\_file: pathname of file to write.

This mode reads lines from an ASCII file, find first two consecutive space-delimited words that can be a lat/lon, and removes the record if that lat/lon is not within the area of the depictor file. To be a lat/lon, a word must represent a floating point or integer number. This mode can perform units conversions on lats and lons. Validity of number as a lat/lon is checked after the units conversion. The usage is as follows:

```
bcdProc i input_file depictor output_file {c1} {latmult} {lonmult}
```

i: the letter `i'  
input\_file: pathname of file to read.  
depictor: pathname of depictor file (.sup file) to clip with.  
output\_file: pathname of file to write.  
c1 (optional): Number of characters to skip in each line of text before trying to find lats and lons. Defaults to 0.  
latmult (optional): Units conversion for latitudes, defaults to 1.  
lonmult (optional): Units conversion for latitudes, defaults to 1.

This mode reads a bcd or bcx file that may not have records truncated to the standard number of points (currently 500) and writes out a file with records that are truncated in that manner. The usage is as follows:

```
bcdProc s {x} input_file output_file
```

s: the letter `s'  
x (optional): if the letter `x' is present, assume .bcx file.  
input\_file: pathname of file to read.  
depictor: pathname of depictor file (.sup file) to clip with.  
output\_file: pathname of file to write.

For the mode that converts an ASCII file to a bcd file, the usage is as follows:

```
bcdProc b {x} input_file output_file
```

b: the letter `b'

x (optional): if the letter `x' is present, assume .bcx file.

input\_file: pathname of file to read.

output\_file: pathname of file to write.

For a bcx file, the ASCII would look like this:

```
label_string
```

```
lat lon
```

```
lat lon
```

```
:
```

```
:
```

```
label_string
```

```
lat lon
```

```
lat lon
```

```
:
```

```
:
```

For a bcd file, the format would look like this:

```
lat lon
```

```
lat lon
```

```
:
```

```
:
```

```
blank_line
```

```
lat lon
```

```
lat lon
```

```
:
```

```
:
```

```
blank_line
```

All lats and lons are decimal degrees, east and north positive. Sequential lines containing lats and lons are treated as a single linked vector. The blank line/ label causes a new linked vector to start. Any number of linked vectors may be included in a file, and they can be any arbitrary length.

---

**Author: Jim Ramer**  
**Last update: 15 Oct 02**

## fileMover

fileMover is a standalone program meant to facilitate moving files into the localization data set. For ASCII files it will always be certain that the resulting file terminates with a newline. The usage is as follows:

```
fileMover {-123} {+123} \  
          {c} {b} {a} {z} {x} {e} {i} {o} {n} {d} {k} {s} {u} \  
          input_file output_file
```

The optional literal single character flags can be in any order among themselves, but must occur before the ``input_file'` argument.

For the `'-123'` and `'+123'` arguments, the `'-'` and `'+'` are literal, and the 123 can be any number of numeric digits.

The default behavior of this program is to copy the ASCII file designated by the argument ``input_file'` to the path designated in the argument ``output_file'`, replacing the contents of the output file. Zero length input files are treated as if the file did not exist.

The default replacement behavior can be changed in one of two ways. As one might expect, it can be changed by a command line argument. Furthermore, if the input file has as its first line ``#append'`, the file will normally append regardless of what the command line arguments say. If the input file has as its first line ``#replace'`, the file will normally replace the output file regardless of what the command line arguments say. There is a command line argument that suppresses this feature.

If the argument ``input_file'` is blank, missing, or a literal `'-'`, the program will take input from stdin.

Similarly, if the argument ``output_file'` is blank, missing, or a literal `'-'`, the program will write to stdout. If the argument ``output_file'` is a literal ``~'`, it will write to stderr. A value of ``='` for the `output_file` argument will cause the output file to have the same file name as the input file but in the current working directory. Finally, a value of ``='` for the output file will do basically the same thing as ``='`, except it will strip any leading `*-` or `*--` off of the output file name.

The `'-123'` and `'+123'` allow one to select parts of a partitioned file. A partitioned file is any file that has lines in it beginning with ``#partition'`. All lines up to the first occurrence are considered to make up partition 1, all lines up to the next occurrence are considered to make up partition 2, and so on. The numeric digits list the partitions to include. A partitioned file that has no data in the selected partitions is considered empty. If the argument used has the leading `'-'`, then a non-partitioned file will be considered empty. The leading `'+'` results in the entire contents of a non-partitioned file being used. By default partitioning is not considered, which would be sort

of like '+123456789'.

Here is the meaning of the rest of the single character literal flags.

- c Strip comments. Comment is anything from a // to the end of the line, and any line that starts with #, except for #include.
- b Strip blank lines. Any line that is empty or all spaces is considered a blank line. If stripping comments, a line that is all comments is considered blank.
- a Append the input file to the output file if it already exists; default behavior is to replace the output file. Normally, if '#append' or '#replace' occurs as the first line of the file, that will override the behavior selected by the presence or absence of this argument.
- z Treat the file as a binary file. This means the c and b flags are ignored, will not try to add a newline at the end of the file if missing.
- x Do not use the presence of '#append' or '#replace' as the first line of the file to determine whether to append or replace.
- e By default, empty (zero length) input files are completely ignored; this will cause them to be processed.
- i Print the path of the input file to standard output.
- o Print the path of the output file to standard output.
- d Write message about any exceptional conditions to standard output.
- n No-op: do not actually move any file data, just process the arguments and print any diagnostics.
- k Treat each line for output as a vertical-bar-delimited key file entry. The first vertical-bar-delimited field on the line is treated as the key. If there are redundant key values, keep only the last entry.
- s Same as k, but with a space as the delimiter.
- u Remove any non-unique lines. Leading and trailing spaces are ignored for determining uniqueness.

---

**Author: Jim Ramer**  
**Last update: 28 Jun 04**

## GELTtest

The program GELTtest is a standalone program that can perform a unit test on the GeoEntityLookupTable class and can filter the contents of ascii location files based on a geographic entity lookup table.

The program allows the user four different methods of operation which are summarized here and explained in detail below:

- Unit test of the GeoEntityLookupTable class.
- Unit test of the GeoEntityLookupTable class using entity ids.
- Keep locations within an entity.
- Keep locations not within an entity.

User note 1:

The ascii location files used here must contain one location per line in the file which have consecutive space delimited fields that can be interpreted as a latitude and longitude.

User note 2:

When doing a unit test, one point results in describing that point, two points results in describing the area to the right of that line, and three or more points results in describing the area inside that polygon. No points describes the whole table.

For the unit test mode, the usage is as follows:

```
GELTtest t gelt_name {p} {x} {c} {w} {v} {g} {G} {lat lon} {lat lon} ...
```

t: The letter `t'.

gelt\_name: Path name to geographic entity lookup table with no extensions;  
uses InfoFileServer with this path.

p (optional): Describe portions of areas.

x (optional): Use keyword `extreme' in describing portions of areas.

c (optional): Most liberal use keyword `central' in describing portions of areas.

w (optional): Returns all of the text associated with each entity instead of just the first field.

v (optional): Also print out size, index, and location of each entity.

g (optional): Treat each different portion of an entity as a separate virtual entity.

G (optional): Treat each different grid point of an entity as a separate virtual entity.

lat: Latitude of a point to describe.

lon: Longitude of a point to describe.

For the unit test mode that uses entity ids, the usage is as follows:

```
GELTtest t gelt_name i field {p} {x} {c} {w} {v} {g} {G} id {id} ...
```

t: The letter `t'.  
gelt\_name: Path name to geographic entity lookup table with no extensions;  
          uses InfoFileServer with this path.  
i: The letter `i'.  
field: Field in table where the ids provided can be found.  
p (optional): Describe portions of areas.  
x (optional): Use keyword `extreme' in describing portions of areas.  
c (optional): Most liberal use keyword `central' in describing portions  
          of areas.  
w (optional): Returns all of the text associated with each entity instead of  
          just the first field.  
v (optional): Also print out size, index, and location of each entity.  
g (optional): Treat each different portion of an entity as a separate  
          virtual entity.  
G (optional): Treat each different grid point of an entity as a separate  
          virtual entity.  
id: Entity id to activate in the table.

For the mode that keeps locations within an entity, the usage is as follows:

```
GELTtest a in_file gelt_name out_file entity_id
```

a: The letter `a'.  
in\_file: Input file with location information.  
gelt\_name: Path name to geographic entity lookup table with no extensions;  
          uses InfoFileServer with this path.  
out\_file: File to write filtered location information to.  
entity\_name: First vertical bar delimited field in the identifier of the  
          geographic entity to compare with (whole identifier if no  
          delimiters).

For the mode that keeps locations not within an entity, the usage is as  
follows:

```
GELTtest i in_file gelt_name out_file entity_id
```

i: The letter `i'.  
in\_file: Input file with location information.  
gelt\_name: Path name to geographic entity lookup table with no extensions;  
          uses InfoFileServer with this path.  
out\_file: File to write filtered location information to.  
entity\_name: First vertical bar delimited field in the identifier of the  
          geographic entity to compare with (whole identifier if no  
          delimiters).

---

**Author: Jim Ramer**  
**Last update: 8 Jul 99**

## image\_mask

The program `image_mask` is a standalone program that can perform very simple operations on flat files that represent image data.

The program allows the user four different methods of operation which are summarized here and explained in detail below:

- Null out pixels in an image based on values in another image.
- Null out values in an image that have too few like neighbors.
- Null out values in an image whose total count in the image is too few.
- For null pixels, assign value from neighbor if non-null in another image.
- Null out values in an image whose fraction of edge pixels is too large.
- Null out values whose total count is too small compared to another image.

User note 1:

In all cases, the image file gets rewritten.

User note 2:

The type of an image concerns how many bytes per pixel it has. For this, a ``b'` means a byte image (one byte per pixel), a ``w'` means a word image (two bytes per pixel), and an ``i'` mean an integer image (four bytes per pixel).

User note 3:

For all usage modes, an optional literal ``e'` as the first argument will result in the endian state for word images to be verified and adjusted if needed, based on the assumption that total variability in the data will be greatest in the least significant byte.

For the mode that nulls out pixels based on values in another image, the usage is as follows:

```
image_mask m image_type image_file mask_type mask_file {null} {n} {min} {max}
```

`m`: The letter ``m'`.

`image_type`: Type of image file (see note 2).

`image_file`: Pathname of the image file.

`mask_type`: Type of mask file (see note 2).

`mask_file`: Pathname of the mask file.

`null` (optional): Value that is used as the null value to place into the image file (defaults to zero).

`n` (optional): By default, pixels are nulled out if the corresponding value in the mask image is from ``min'` to ``max'`. If this literal `n` is present, pixels are nulled out if the mask value is outside this range.

`min` (optional): Lowest value in the mask image that triggers insertion of a null into the main image file (defaults to zero).

`max` (optional): Highest value in the mask image that triggers insertion of a null into the main image file (defaults to zero).

Note that the image and the mask must have the same number of pixels, but can have different types.

For the mode that nulls out pixels that have too few like neighbors, the usage is as follows:

```
image_mask n image_type image_file nx ny {min} {null}
```

n: The letter `n'.

image\_type: Type of image file (see note 2).

image\_file: Pathname of the image file.

nx: Inner dimension of the image.

ny: Outer dimension of the image.

min (optional): Less than this many like-value neighbors trigger insertion of a null into a pixel in the image file (defaults to one).

null (optional): Value that is used as the null value to place into the image file (defaults to zero).

Note that each pixel can potentially have eight neighbors.

For the mode that nulls out pixels that have too few like values in the entire image, the usage is as follows:

```
image_mask c image_type image_file {min} {null}
```

c: The letter `c'.

image\_type: Type of image file (see note 2).

image\_file: Pathname of the image file.

min (optional): Less than this many like values in the whole image triggers insertion of a null into a pixel (defaults to one).

null (optional): Value that is used as the null value to place into the image file (defaults to zero).

For the mode that assigns neighboring values to null pixels if they are non-null in another image, the usage is as follows:

```
image_mask f image_type image_file nx ny mask_type mask_file {null} {min} {max}
```

f: The letter `f'.

image\_type: Type of image file (see note 2).

image\_file: Pathname of the image file.

nx: Inner dimension of the images.

ny: Outer dimension of the images.

mask\_type: Type of mask file (see note 2).

mask\_file: Pathname of the mask file.

null (optional): Null value in image file (defaults to zero).

min (optional): Lowest value in the mask image that does not allow setting data values in the main image file (defaults to zero).

max (optional): Highest value in the mask image that does not allow setting data values in the main image file (defaults to zero).

Note that the image and the mask must have the same number of pixels, but can have different types.

For the mode that nulls out values in an image whose fraction of edge pixels is too large in the entire image, the usage is as follows:

```
image_mask s image_type image_file nx ny {pct} {null}
```

s: The letter `s'.

image\_type: Type of image file (see note 2).

image\_file: Pathname of the image file.

nx: Inner dimension of the image.

ny: Outer dimension of the image.

pct (optional): Threshold percentage (defaults to 10).

null (optional): Null value in image file (defaults to zero).

Note that an edge pixel is defined as one that has any of its eight neighbors with a different value.

For the mode that nulls out all pixels of a value when its total count in the image is too small compared to the total count in the mask, the usage is as follows:

```
image_mask p image_type image_file mask_type mask_file {pct} {null}
```

p: The letter `p'.

image\_type: Type of image file (see note 2).

image\_file: Pathname of the image file.

mask\_type: Type of mask file (see note 2).

mask\_file: Pathname of the mask file.

pct (optional): Threshold percentage (defaults to 5). All pixels of a given value in the image are nulled out if their total count is less than this percentage of the total count in the mask image.

null (optional): Value that is used as the null value to place into the image file (defaults to zero).

---

**Author: Jim Ramer**  
**Last update: 30 Oct 02**

## initCdlTemplate

initCdlTemplate is a standalone program that is used to write geographically dependent and topographic information into a template NetCDF file. Specifically, it writes values to a standard set of global attributes that define the geographic characteristics of all WFO-advanced plan-view grid and image file. It also writes a topography grid, a grid spacing grid(s), and a coriolis parameter grid into the template file.

If the user does not supply a topography grid, it will not attempt to write one. If variable space for the grid spacing grid(s), coriolis parameter, or topography (when supplied) does not exist, the program will continue and will warn the user about those fields it could not write.

The usage is as follows:

```
initCdlTemplate templateFile depictorFile {topoFile}
```

templateFile: Full path to the NetCDF template file being written to.

depictorFile: Full or abbreviated path name of the depictorFile that describes the geographic characteristics of the grid.

topoFile (optional): Full path to a file containing topography for the grid. File should be an ascii file, one value per line, scanning left to right, bottom to top.

User Note: This program is not meant to create variable space or define the existence of the global attributes in question. It is assumed that these items are all defined in the cdl file from which the template file was created with an `ncgen' command.

---

**Author: Jim Ramer**  
**Last update: 6 Jun 98**

## keyMunge

keyMunge is a standalone program meant to change key values in key files in an automated fashion; the usage is as follows:

```
keyMunge {c} main_file key1 key2 keyAdd output_file {term}
```

The `c` is an optional literal flag that will cause keys to be output in a compressed base 32 format. 'main\_file' is the file being edited, and 'output\_file' is the file to write the results to. 'key1' and 'key2' are the range of keys which are changed in the 'main\_file'. 'keyAdd' is the value that is added to each key in the 'main\_file' to produce what is written out in the 'output\_file'. A key is defined as any consecutive string of digits, the resulting value of which falls within the range specified.

The optional 'term' is an additional character, besides the normal comment sequences of "//" and "#", which will cause processing of the line for additional keys to terminate.

---

**Author: Jim Ramer**  
**Last update: 21 Jul 99**

# Grid Key Tables

makeGridKeyTables is a standalone program that produces the data and depictable key entries for gridded data, using as input the contents of five tables: gridSourceTable.txt, dataLevelTypeTable.txt, gridPlaneTable.txt, dataFieldTable.txt, and virtualFieldTable.txt. The files created are gridDataKeys.txt and gridDepictKeys.txt. For a complete description of how these grid tables work, see [gridTables](#).

Beyond this primary function makeGridKeyTables can also automatically generate multi-loads based on gridded data. An additional input file it uses to do this is comparisonFields.txt, and the additional output file it creates to do this is gridMultiLoadKeys.txt. For more details about this, see [families](#).

The usage for makeGridKeyTables is as follows:

```
makeGridKeyTables {v} {x} {t} {u} {c}
```

v (optional):

Causes itemized list of all raw data fields identified to be written to standard output.

x (optional):

Completely expands out all key entries. Normally, many cross section depict keys are abbreviated because they are so much like many other cross section keys.

t (optional):

Terse form, which is used in the localization. No titles in data keys and no second legend for depict keys.

u (optional):

Do a unit test on the GridTableServer object before creating the keys.

c (optional):

Causes keys to be compressed into base 32 format.

User note: In order to get meaningful titles for cross section planes, one needs to have run the command `testGridKeyServer G', which also depends on having access to depictor files for display scales.

---

**Author: Jim Ramer**

**Last update: 21 Mar 97**

## maksubgrid

maksubgrid is a standalone program that is meant to create a geographic information file that represents a portion of a grid. The usage is as follows:

```
maksubgrid base_geo base_nx base_ny clip_geo clip_nx clip_ny \  
           { offset_x offset_x } output_geo {IngestCenter} {o}
```

base\_geo: geographic information file that defines the base grid.

base\_nx: x dimension of the base grid.

base\_ny: y dimension of the base grid.

clip\_geo: output clipped grid approximately centered over the area defined by this geographic information file.

clip\_nx: x dimension of the output clipped grid.

clip\_ny: y dimension of the output clipped grid.

offset\_x (optional): number of grid points to shift in the x direction.

offset\_y (optional): number of grid points to shift in the y direction.

output\_geo: name of the geographic information file to write that describes the clipped grid.

IngestCenter: Must be the 10th argument. With this option, it use IngestCenter.dat

to find the center point instead of from base\_geo(see 1st argument) file.

o : Must be the last argument. With this option, the clipped area can be {partly} outside of the original data area.

The file output\_geo will always describe the same projection as base\_geo, and will describe a different area that fits within the area of base\_geo. clip\_geo can be any arbitrary projection; it is used merely to provide the area around the clipped grid is centered. It must be true that clip\_nx <= base\_nx and clip\_ny <= base\_ny. The optional shift is applied after the normal default determination of the output area, but the shift is truncated so that the area of output\_geo still falls within the area of base\_geo.

---

**Author: Jim Ramer**

**Last update: 11 Mar 08 by Wen Kwock**

## Creating a Geographic Information File

This program is a stand-alone program for creating a geographic information file ("sup" file).

The program allows the user ten different methods by which to define the area covered by the geographic information file, a report of the percent overlap between two .sup files, and a way to recreate the arguments which created a file. These are summarized here and explained in detail below.

1. Entire useful area of the projection.
2. Opposing lat/lon corners are specified.
3. Ranges of Cartesian coordinates are specified.
4. Offset distances from a point are specified.
5. Corners of a grid defined by reference corner plus grid spacing.
6. Area defined by an ASCII map points file.
7. Area defined by a .bcd map points file.
8. Move (but not resize) the domain of an existing file so its border does not overlap the border of another.
9. Expand the domain of an existing file to include another.
10. Restrict the domain of an existing file to the intersection with another.
11. Print the percent overlap between two files.
12. Regurgitate a set of argument lists that can be used to recreate an existing file.

User note 1:

Where projection indices are referred to, here is their meaning:

- 1 = STEREOGRAPHIC
- 2 = ORTHOGRAPHIC
- 3 = LAMBERT\_CONFORMAL (rotation is second standard parallel)
- 4 = AZIMUTHAL\_EQUAL\_AREA
- 5 = GNOMONIC
- 6 = AZIMUTHAL\_EQUIDISTANT
- 7 = SATELLITE\_VIEW (defaults to geostationary height)
- 8 = CYLINDRICAL\_EQUIDISTANT
- 9 = MERCATOR
- 10 = MOLLWEIDE
- 16 = AZIMUTH\_RANGE
- 17 = RADAR\_AZ\_RAN

User note 2:

Anywhere the following list of arguments is found:

```
proj_idx center_lat center_lon rotation {orbit_hgt}
```

the user may replace this with path name to an existing geo file, from which the program will extract these arguments.

User note 3:

Anywhere a path name to an existing geo file is needed, it can be either

the full path name or the completion of \$GEO\_DATA/\*.sup

User note 4:

The optional orbit\_hgt argument can be used only when the SATELLITE\_VIEW projection (projection 7) is being used. Defaults to a geostationary height. This projection assumes a spherical earth, but when defining a sector in terms of correct lat/lon corners, the depiction will be quite accurate unless it is covering a large area.

User note 5:

Use the CYLINDRICAL\_EQUIDISTANT projection to define lat/lon based areas. The arbitrary x and y coordinates for this projection correspond to degrees of longitude and latitude. When covering the whole globe, use the x option with -180 180 -90 90 as your ranges of x/y coordinates. The center\_lon would then be 180 degrees from the seam. Generally, the center\_lat, center\_lon, and rotation parameters for this projection will always be 0 0 0 unless one is defining an area that spans the date line.

User note 6:

The user should keep in mind that the center\_lat and center\_lon parameters refer to the center/origin/tangent point of the projection, which is not necessarily the same as the center point of the area one is trying to cover with the range of x/ys or lat/lons.

1. For the whole projection mode, the usage is as follows:

```
maksuparg proj_idx center_lat center_lon rotation {orbit_hgt} output_file
```

proj\_idx: projection index  
center\_lat: central latitude or latitude of tangent point  
center\_lon: central longitude or longitude of tangent point  
rotation: projection rotation  
orbit\_hgt (optional): height of orbit (km) for SATELLITE\_VIEW only  
output\_file: name of geographic information file to write

2. For the lat/lon corners mode, the usage is as follows:

```
maksuparg proj_idx center_lat center_lon rotation {orbit_hgt} output_file  
1 lat_1 lon_1 lat_2 lon_2
```

proj\_idx: projection index  
center\_lat: central latitude or latitude of tangent point  
center\_lon: central longitude or longitude of tangent point  
rotation: projection rotation  
orbit\_hgt (optional): height of orbit (km) for SATELLITE\_VIEW only  
output\_file: name of geographic information file to write  
1: the character "1"  
lat\_1: latitude of the first corner  
lon\_1: longitude of the first corner  
lat\_2: latitude of the opposing corner  
lon\_2: longitude of the opposing corner

3. For the Cartesian range mode, the usage is as follows:

```
maksuparg proj_idx center_lat center_lon rotation {orbit_hgt} output_file
          x min_x max_x min_y max_y
```

proj\_idx: projection index  
center\_lat: central latitude or latitude of tangent point  
center\_lon: central longitude or longitude of tangent point  
rotation: projection rotation  
orbit\_hgt (optional): height of orbit (km) for SATELLITE\_VIEW only  
output\_file: name of geographic information file to write  
x: the character "x"  
x\_min: smallest x value  
x\_max: largest x value  
y\_min: smallest y value  
y\_max: largest y value

4. For the offset distance mode, the usage is as follows:

```
maksuparg proj_idx center_lat center_lon rotation {orbit_hgt} output_file
          o base_lat base_lon left right bottom top
```

proj\_idx: projection index  
center\_lat: central latitude or latitude of tangent point  
center\_lon: central longitude or longitude of tangent point  
rotation: projection rotation  
orbit\_hgt (optional): height of orbit (km) for SATELLITE\_VIEW only  
output\_file: name of geographic information file to write  
o: the character "o"  
base\_lat: latitude of the base point  
base\_lon: longitude of the base point  
left: distance in km from base point to the left edge  
right: distance in km from base point to the right edge  
bottom: distance in km from base point to the bottom edge  
top: distance in km from base point to the top edge

In this case, if left=right and bottom=top, then the user can just enter the left/right and bottom/top.

5. For the grid definition mode, the usage is as follows:

```
maksuparg proj_idx center_lat center_lon rotation {orbit_hgt} output_file
          g def_lat def_lon base_lat base_lon dx dy nx ny {earth}
```

proj\_idx: projection index  
center\_lat: central latitude or latitude of tangent point  
center\_lon: central longitude or longitude of tangent point  
rotation: projection rotation  
orbit\_hgt (optional): height of orbit (km) for SATELLITE\_VIEW only  
output\_file: name of geographic information file to write  
g: the character "g"  
def\_lat: latitude of point that defines distance scaling  
def\_lon: longitude of point that defines distance scaling  
base\_lat: latitude of the reference corner

base\_lon: longitude of the reference corner  
dx: x grid spacing in km  
dy: y grid spacing in km  
nx: number of grid points in the x direction  
ny: number of grid points in the y direction  
earth (optional): radius of earth to use in km, defaults to 6371.

6. For the ASCII map points file mode, the usage is as follows:

```
maksuparg proj_idx center_lat center_lon rotation {orbit_hgt} output_file  
a map_file extra minwidth maxwidth
```

proj\_idx: projection index  
center\_lat: central latitude or latitude of tangent point  
center\_lon: central longitude or longitude of tangent point  
rotation: projection rotation  
orbit\_hgt (optional): height of orbit (km) for SATELLITE\_VIEW only  
output\_file: name of geographic information file to write  
a: the character "a"  
map\_file: the name of the file containing map points in ASCII format  
extra (optional): extra margin in km added around the area of the points  
minwidth (optional): minimum width of the area in km  
maxwidth (optional): maximum width of the area in km

If the "a" is upper case, this mode will not square off the area.

7. For the binary map points file mode, the usage is as follows:

```
maksuparg proj_idx center_lat center_lon rotation {orbit_hgt} output_file  
b map_file extra minwidth maxwidth
```

proj\_idx: projection index  
center\_lat: central latitude or latitude of tangent point  
center\_lon: central longitude or longitude of tangent point  
rotation: projection rotation  
orbit\_hgt (optional): height of orbit (km) for SATELLITE\_VIEW only  
output\_file: name of geographic information file to write  
b: the character "b"  
map\_file: the name of the file contain map points in binary format  
extra (optional): extra margin in km added around the area of the points  
minwidth (optional): minimum width of the area in km  
maxwidth (optional): maximum width of the area in km

If the "b" is upper case, this mode will not square off the area.

8. For the mode which moves a domain to eliminate border overlap with another file, the usage is as follows:

```
maksuparg input_file mod_file n output_file
```

input\_file: path to existing file whose domain to move  
mod\_file: path to existing modifier geo file  
n: the character "n"  
output\_file: name of geographic information file to write

9. For the mode which expands the domain of an existing file to include another, the usage is as follows:

```
maksuparg input_file mod_file u output_file
```

input\_file: path to existing file whose domain to expand  
mod\_file: path to existing modifier geo file  
u: the character "u"  
output\_file: name of geographic information file to write

10. For the mode which restricts the domain of an existing file to the intersection with another, the usage is as follows:

```
maksuparg input_file mod_file i output_file
```

input\_file: path to existing file whose domain to restrict  
mod\_file: path to existing modifier geo file  
i: the character "i"  
output\_file: name of geographic information file to write

11. For the mode which prints the percent overlap between two files, (relative to the coverage of the first file) the usage is as follows:

```
maksuparg reference_file in_question_file p
```

reference\_file: path to reference file  
in\_question\_file: path to file whose relative size you wish to know  
p: the character "p"

Note: If the overlap is less than 1% but greater than zero, 1 is returned. 0 is returned only for a no-overlap situation.

12. For the arguments recreation mode, the usage is as follows:

```
maksuparg file {lat lon}
```

file: geographic information file whose creation arguments to retrieve  
lat (optional): center lat for offset distance mode  
lon (optional): center lon for offset distance mode

---

**Author: Jim Ramer**  
**Last update: 6 Apr 04**

## makthermo

makthermo is a standalone program that is used for making thermodynamic depicators. The program operates in two modes; one where the corners of the diagram are given in temperature and pressure, and one where the corners are given in the arbitrary x/y coordinates of the diagram.

### User note 1:

Where diagram indicies are refered to, here is their meaning:

1 = P\_T\_LINEAR  
2 = SKEW\_T  
3 = STUVE  
4 = P\_LOG\_T\_LIN

### User note 2:

Temperatures can be input in either C or K, pressures can be input in either pascals or millibars. Aspect ratios are in units of degK/pas, a value of 0.00033 is nominal for a run of the mill skewT with an origin at 1000mb.

### User note 3:

This routine cannot add the optional lat and lon that can be included in a thermodynamic depicator file. However, a thermodynamic depicator file is just an ascii file, so a user can append a line with a latitude, then append a line with a longitude.

For the temperature and pressure corners mode, the usage is as follows:

```
makthermo diag_idx p_origin t_origin aspect thermo_dep t \  
          press_1 temp_1 press_2 temp_2
```

diag\_idx: diagram index.

p\_origin: pressure where arbitrary x and y are both 0.

t\_origin: temperature where arbitrary x and y are both 0.

aspect: aspect ratio of the diagram.

thermo\_dep: the name of the thermodynamic depicator file to write.

t: the letter 't'.

press\_1: pressure of one corner.

temp\_1: temperature of one corner.

press\_2: pressure of opposite corner.

temp\_2: temperature of opposite corner.

For the cartesian corners mode, the usage is as follows:

```
makthermo diag_idx p_origin t_origin aspect thermo_dep x \  
          min_x max_x min_y max_y
```

diag\_idx: diagram index.

p\_origin: pressure where arbitrary x and y are both 0.

t\_origin: temperature where arbitrary x and y are both 0.  
aspect: aspect ratio of the diagram.  
thermo\_dep: the name of the thermodynamic depictor file to write.  
x: the letter `x'.  
min\_x: minimum x value on the diagram.  
max\_x: maximum x value on the diagram.  
min\_y: minimum y value on the diagram.  
max\_y: maximum y value on the diagram.

---

**Author: Jim Ramer**  
**Last update: 6 Jun 98**

## makxsect

makxsect is a standalone program that is used for making cross section depicter files, which can be either time-height or space height.

### User note 1:

The default vertical representation for all cross sections is a log pressure vertical coordinate with 1050mb at the bottom and 150mb at the top.

### User note 2:

Where projection indices are referred to, here is their meaning:

- 1 = STEREOGRAPHIC
- 2 = ORTHOGRAPHIC
- 3 = LAMBERT\_CONFORMAL
- 4 = AZIMUTHAL\_EQUAL\_AREA
- 5 = GNOMONIC
- 6 = AZIMUTHAL\_EQUIDISTANT
- 7 = SATELLITE\_VIEW
- 8 = CYLINDRICAL\_EQUIDISTANT
- 9 = MERCATOR
- 10 = MOLLWEIDE
- 16 = AZIMUTH\_RANGE
- 17 = RADAR\_AZ\_RAN

The most common usage for creating time-height cross sections is as follows:

```
makxsect xsect_dep seq_num lat lon
```

xsect\_dep: The name of the xsect depicter file to write.

seq\_num: Two time-height depictors with different sequence numbers cannot overlay even if their vertical representations are the same.

lat: Latitude of the time-height cross section.

lon: Longitude of the time-height cross section.

The most common usage for creating spatial cross sections is as follows:

```
makxsect xsect_dep 6 lat lon lat lon {lat lon} ...
```

xsect\_dep: The name of the xsect depicter file to write.

6: The number `6', which means baseline is computed using an azimuthal equidistant projection.

lat (optional): Latitude of a baseline point (must be at least 2).

lon (optional): Longitude of a baseline point (must be at least 2).

The complete usage allows the user control of the vertical representation and the projection upon which spatial baselines are computed. The complete usage description is as follows:

```
makxsect xsect_dep geo_proj {c cen_lat cen_lon} {vcoord bottom_p top_p} \  
    {lat lon} {lat lon} {lat lon} ...
```

xsect\_dep: The name of the xsect depictor file to write.  
geo\_proj: Either a sequence number, projection index, or a geographic  
depictor file.  
c (optional): The letter `c', which specifies that center lat and lon follow.  
cen\_lat (optional): Center latitude of projection cross section is based on.  
cen\_lon (optional): Center longitude of projection cross section is based on.  
vcoord (optional): Either a 'p' for log p vertical coordinate, or 'P' for a  
linear p vertical coordinate.  
bottom\_p (optional): Pressure of the bottom of the cross section.  
top\_p (optional): Pressure of the top of the cross section.  
lat (optional): Latitude of one baseline point.  
lon (optional): Longitude of one baseline point.

If one provides a geographic depictor file and no baseline points, this routine will provide a cross section baseline with the lower left and upper right corners of the depictor file as baseline points. One can specify the geographic depictor that the baseline is computed with either by directly supplying it or by specifying a projection index and a center point. Supplying only one baseline point is a signal to create a time-height depictor.

---

**Author: Jim Ramer**  
**Last update: 6 Jun 98**

# newGELTmaker

## 1) Introduction

newGELTmaker is a standalone program that is used to create geographic entity lookup tables (GELTs). GELTs are central to how warnGen works; it is how warnGen can find out what counties, cities, or zones are in an arbitrary geographic area. newGELTmaker has the same basic functionality as the older program makeGeoTables, but is more powerful and more flexible. The program makeGeoTables is obsolete and has been removed from AWIPS.

The usage of newGELTmaker is as follows

```
newGELTmaker {-v} gelt_script_file
```

Most of this document will be devoted to describing how GELT script files work. When the -v is present, newGELTmaker will just list the pathname of one of the files that the GELT script will try to write, followed by the pathnames of all the files that the GELT script will try to read. Otherwise it will try to generate one GELT based on the contents of the GELT script.

## 2) Geographic Entity Lookup Tables

At this point it is helpful to the discussion to say something more about geographic entity lookup tables (GELTs).

A GELT is a table that allows the client to get information about which geographic entities exist at some point or within some area. A geographic entity might be a city, a county, or a forecast zone, for example. A GELT is composed of six parts, each being a file with the same path name except for the extension. The file with a `.gelt` extension contains a description of the grid on which the table is based and a list of the rest of the files in the GELT. Because the `.gelt` file contains a list of the rest of the files, one can alias a GELT by just creating a symbolic link for the `.gelt` file. The file with a `.id` extension contains the earth coordinates and ASCII identifiers associated with each geographic entity in the table. The file with the `.entity` extension lists each contiguous area in the table and maps each contiguous area to one or more identifiers in the `.id` file. The file with the `.table` extension is a grid of 2-byte integers, each a pointer to an item in the `.entity` file. The files with the extensions `.NS` and `.EW` contain information about how far north, south, east, or west within an entity a given point is. When created, the files that make up a GELT are written to the `localizationDataSets/LLL/` directory.

The `.entity`, `.table`, `.NS`, and `.EW` files contain either binary data or data that are hard to make sense of except by programs, and the format and contents of the `.gelt` file are pretty straightforward. However, the contents of the `.id` files are both human viewable and important enough to understand that they will be described in detail here. To begin the discussion, here are the contents of the first three lines of an arbitrary `.id` file that is part of a GELT:

```
93
1  40.714 -103.107 a    1  LOGAN           |ne CO| COC075 | BOU
2  40.871 -102.354 a    2  SEDGWICK        |ne CO| COC115 | BOU
```

Should the reader be curious, this is from the warning county table of a BOU localization, but that is not important for now. The first line is always a count of the total number of entries in the file. The first 5 fields on each entry line are space delimited. The first and fifth field are sequence numbers that should be considered internal to the GELT software. The second and third fields are a latitude and longitude. The fourth ('a' in this example) is the entity type. A type of 'p' means the entity is a point entity represented by one grid point. The other types mean that the entity is an area entity represented by multiple grid points. For area entities, the exact type corresponds to how the centroid used for computing north, south, east, and west was calculated. The 'a' as in the example means that the centroid is area weighted. An 'A' means the centroid was an arbitrary point supplied by the user. An 'm' means the centroid is the point within the entity furthest from any boundary. The user should note that the lat/lon in the ID file is always the same as the 'm' style centroid, not necessarily the same as the north-south-east-west centroid.

After the first five space-delimited fields is the arbitrary text associated with each entity. The software that reads GELTs recognizes the vertical bar as a field delimiter. Later on as we discuss the complexities of output text formatting, keep in mind that the goal of that is precise control over the creation of this arbitrary text.

### 3) Input data sets

As tables are built, there are two main things that go into the creation of the information concerning each entity: the text associated with it and its areal coverage. Sometimes the information that defines both the text and the areal coverage will come from the same file, sometimes it will come from different files.

There are three kinds of files from which the information about the text associated with each entity can be obtained: cities files, ID files, and shape files. The ID file type has exactly the same format as described in the previous section. Shape files are the public domain format of files used by the ArcInfo GIS system, and AWIPS uses them as its primary format of cartographic data exchange. The default set of shape files for AWIPS lives in the directory localization/nationalData/ and comes in triplets with extensions of .shp[.z], .shx, and .dbf. The AWIPS software that reads shape files can decompress them on the fly so they often exist in a compressed state. To learn about how to query shape files for their list of attributes, please see [shp2bcd](#).

Currently, the default configuration recognizes only one file that is in the cities format, /awips/fxa/data/CitiesInfo.txt. A site may also optionally supply an override file called LocalCitiesInfo.txt. Though referred to as a cities file, this file may have other kinds of locations in it (national parks, military bases, etc.) that are not necessarily municipalities. A cities file is a plain ASCII file with one entry per line. Here is the format of each line:

```
latitude longitude goodness state name|n?
```

The fields 'latitude' and 'longitude' are, of course, the location of the place in question. The field 'goodness' is used for progressive disclosure and so is not usually applicable for building GELTs. The field 'state' normally just contains a two character postal abbreviation. Optionally, additional text can follow the postal abbreviation in the 'state' field that allows two cities with

the same name in the same state to nonetheless be identified as unique. Unlike other fields, the `name` field can have any number of spaces in it. If two or more entries occur that have identical `name` and `state` fields, only the last entry is used. If the `|n?` is present, then this city is important to the warning function. If this is not present, then this entry will usually not be used as an input to a GELT. If n is 1 it is a major city, 2 represents a location of average importance, and 3 is for a minor location. Occasionally, n will be `01` which is used when a city is actually a city/county hybrid. The `?`, if present, is the character `=`, `+`, or `~`, and means that this city is too large in area to be considered a point. The `+` means use an area-weighted centroid, `~` means use the point farthest from any border as the centroid, and `=` means use the supplied point as the centroid. The centroid is a point used to calculate what is north, south, east, or west in the city. The `+`, `=` and `~` correspond to the centroid types `a`, `A`, and `m` mentioned in the previous section.

The most important thing to remember about configuring the output text is that it is always based on attributes. While the shape files are technically the only file format mentioned here that has attributes, the ID files and cities files have virtual attributes. For the ID files, the virtual attributes that exist are `ZERO`, `ONE`, `TWO`, `THREE`, `FOUR`, `FIVE`, `SIX`, and `SEVEN`. ZERO means the whole associated text string, and ONE through SEVEN refer to the individual vertical bar delimited fields that may exist. If no vertical bars exist, then ZERO and ONE mean the same thing and the rest are undefined. For cities files, the virtual attributes that exist are `RAW`, `STATE`, `SUFFIX`, `NAME`, `LEVEL`, and `CTYPE`. RAW means all text from the `state` field to the end of the line. STATE means the two character postal ID in the `state` field, and SUFFIX refers to any additional characters in that field. NAME is the text of the `name` field, LEVEL is the text of the `n` field, and CTYPE is the text of the `?` field. The fact that newGELTmaker treats all these file types as having attributes (virtual or otherwise) allows one to use the same methodology to create the output text for each file type.

For point entities, the areal coverage is determined by just a single lat/lon point. Thus, the areal coverage of a point entity is usually determined from the same file used to derive its associated text. For area entities, one needs to use either a shape file or a .bcd file to define the areas of the entities. Thus, it is possible to define a GELT with area entities completely from a single shape file. However, if one wants to define a GELT with area entities using information from an ID file or a cities file to provide the associated text, then one must define the areal coverage using a .bcd file or a .shape file.

The final type of input data set is the special override file. Like a bcd file, this kind of file cannot result in adding entities to a GELT, but only can modify entities that are added from other files. Each line in a special override file is as follows:

```
- t lat lon id_string @
```

All the items in a special override file entry must retain this order. However, each item except for the `id\_string` item is optional. The `id\_string` item is used as cross reference to an entry that has already been defined somehow from one of the other file types. If the `-` (literal minus sign) is present, it means do not use a shape file to define the area of the entity; this usually means fall back to using a bcd file to define the area even though a shape is available for the entity. If the `t` is present, it must be one of the single letter centroid type codes just like one finds in an ID file (a, A, m, or p); this allows the user to change how the centroid is defined. If present, the `lat` and

`lon' fields must be latitude and longitude in degrees N and E; this allows the user to supply and alternate location for the entity. This only affects the calculation of what is north, south, east, and west if the A centroid type is in effect, and will affect the actual location in the table if the p centroid type is in effect. The `@' (literal at sign) item, if present, allows grid points in this entity to overwrite grid points that have already been defined.

## 4) GELT script files

GELT script files are plain ASCII files that contain one keyword/value pair on each line. The keyword occurs first on the line, followed by the value, space delimited. If the user wants to preserve multiple spaces in the value, the value can be placed in double quotes. Blank lines are OK, and the software that reads the GELT script files understands C++ style // comments.

There are three main types of keywords, those that pertain to the whole GELT script (global keywords), those that introduce an input file (file keywords), and those that determine how an input file will be used to create the GELT (usage keywords). The global keywords and file keywords may appear anywhere in the GELT script. The usage keywords must appear after the file keyword to which they are meant to apply but before the next file keyword.

### 4.1) Global keywords

geo\_file

Allows the user to specify the name of the depictor file (minus the .sup extension) on which the GELT will be based. This must be supplied for a GELT to be successfully built.

grid\_size

Allows the user to specify the spatial resolution of the GELT. If greater than 50, it is assumed to be the number of grid points in the dimension of the depictor file with the greatest extent. Otherwise, it can be a floating point value that is the grid point resolution in kilometers. This must be supplied for a GELT to be successfully built.

output\_file

Allows the user to specify the name of the GELT file to be created. This is given minus any of the different extensions (.gelt, .table, .entity, .id, .NS, .EW) that make up the parts of the GELT. This must be supplied for a GELT to be successfully built.

supress\_file

If not present, this defaults to the argument of the output\_file keyword plus a **.suppress** extension. This file contains a list of entities for which the creation of positional information will be suppressed. Positional information is what is written into the .NS and .EW files; this is information about how far north/south/east/west one is within the entity. Each line in this file should contain a string referring to one entity, that string being the same as is produced by the **unique\_fmt** and **unique\_attr** keywords. A line ending in an optional **:ns** will suppress only north/south positional information for that entity, and a line ending in an optional **:ew** will suppress only east/west positional information. If the argument supplied to the supress\_file keyword is exactly one character long, this functionality will be disabled. It will not cause a

problem if the file ultimately referred to does not exist or is empty; that just means that no entities will have the creation of positional information suppressed.

#### diag\_dump

Allows the user to specify that additional diagnostics should be sent to stderr. A value of 0 (default) means that no additional diagnostics will appear. A value of 1 means high level diagnostics, 2 means verbose diagnostics, 3 means diagnostics for each entry in all the input files, and 4 means diagnostics for each individual edit of an attribute value.

#### topologic

A value of `true` slightly changes the way areas based on .bcd files are calculated. Sacrifices some accuracy near boundaries in order to be more sure that closed boundaries do not "leak."

#### inside

If present, invokes a strategy for building the GELT that assumes that one and only one entity exists, defined by a .bcd file. The value is the ID string associated with areas inside this entity.

#### outside

If present, invokes a strategy for building the GELT that assumes that one and only one entity exists, defined by a .bcd file. The value is the ID string associated with areas outside this entity.

#### bcd\_file

The argument is the path to a .bcd file to read in. There can be any number of bcd\_file keywords.

#### bcd\_warn

If the value for this keyword is `true`, the user will be warned if a .bcd file cannot be read in.

## 4.2) File keywords

File keywords can be divided into two main types, regular and override. Regular file entries designate files that can be used to add entities to a GELT from that file. Override file entries designate files that can only modify entities created from a regular file. The first three keywords listed here refer to regular files, the last three to override files. The user should note that an override file entry will never do anything unless there is a corresponding `override\_with` (see next section) keyword applied to some regular file.

#### add\_by\_shape

The argument to this keyword is a name of a shape file set (minus the .shp, .shx, or .dbf extensions) that will be used to directly add entities to the GELT.

#### add\_by\_id

The argument to this keyword is a name of an ID file that will be used to directly add entities to the GELT.

add\_by\_cities

The argument to this keyword is the name of a cities file that will be used to directly add entities to the GELT.

special\_override

The argument to this keyword is a name of a special override type of file that will be used to alter the characteristics of entities defined in the GELT using other file types.

shape\_override

The argument to this keyword is a name of a shape file set (minus the .shp, .shx or .dbf extensions) that will be used to define the area of coverage for entities that have been defined using a different file.

id\_override

This file allows one to override the centroid type, point location, and associated text for an existing entity with the information from an ID file.

### **4.3) Usage keywords**

override\_with

This keyword is meaningful only for regular files. The argument to this keyword is the name of an override file that can be applied to this file to change certain characteristics of some entities. One regular file may have any number (including zero) of override files associated with it.

warn

If the value of this keyword is "true", a warning will be output to standard error if this input file cannot be read.

overlay

If the value of this keyword is "true", then entities defined from this file can displace already defined entities. Under normal circumstances, once a grid point has been assigned to an entity, it cannot be reassigned to another. The exception is for point entities, which always will displace grid points associated with existing area entities. When a point entity is located at the same grid point as an already existing point entity, then both point entities will be linked to that grid point.

output\_attr

The value of this keyword is the attribute list for creating the associated text for each entity. It combines with the format string that is given as an argument to the output\_fmt keyword to define the text that eventually appears in the .id file in the GELT for each entity from this file. See [section 5.2](#) for an explanation of attribute lists and format strings.

output\_fmt

The value of this keyword is the format string for creating the associated text for each entity. It combines with the attribute list that is given as an argument to the output\_attr keyword to define the text that eventually appears in the .id file in the GELT for each entity from this file. See [section 5.2](#) for an explanation of attribute lists and format strings.

#### unique\_attr

The value of this keyword is the attribute list for creating the text that identifies each unique ID. If two entries in a file end up having the same value for this resulting text, they will be combined into a single entity. This works in conjunction with the format string given as an argument to the unique\_fmt keyword to control how this text is generated. See [section 5.2](#) for an explanation of attribute lists and format strings.

#### unique\_fmt

The value of this keyword is the format string for creating the text that identifies each unique ID. If two entries in a file end up having the same value for this resulting text, they will be combined into a single entity. This works in conjunction with the attribute list given as an argument to the unique\_attr keyword to control how this text is generated. See [section 5.2](#) for an explanation of attribute lists and format strings.

#### filter\_attr

The value of this keyword is the attribute list for creating the text that allows one to control which entities from the file actually end up being used in the final GELT. The text that results from this must match the regular expression given as an argument to the filter\_exp keyword and not match the argument to the keyword filter\_exp\_not. If no argument is given to either filter\_exp or filter\_exp\_not, then this feature is disabled, and if only one of these is present then only one test is made. This works in conjunction with the format string given as an argument to the filter\_fmt keyword to control how this text is generated. See [section 5.2](#) for an explanation of attribute lists and format strings.

#### filter\_fmt

The value of this keyword is the format string for creating the text that allows one to control which entities from the file actually end up being used in the final GELT. The text that results from this must match the regular expression given as an argument to the filter\_exp keyword and not match the argument to the keyword filter\_exp\_not. If no argument is given to either filter\_exp or filter\_exp\_not, then this feature is disabled, and if only one of these is present then only one test is made. This works in conjunction with the format string given as an argument to the filter\_fmt keyword to control how this text is generated. See [section 5.2](#) for an explanation of attribute lists and format strings.

#### filter\_exp

The presence of this keyword activates a feature that allows one to eliminate individual entities in a file from inclusion in a GELT based on not matching the regular expression given as an argument to this keyword. Only those entities where this regular expression matches with the string resulting from the filter\_attr and filter\_fmt keyword arguments will be included in the GELT. If the filter\_exp\_not keyword is also present then both tests must pass for an entity to be included.

#### filter\_exp\_not

The presence of this keyword activates a feature that allows one to eliminate individual entities in a file from inclusion in a GELT based on matching the regular expression given as an argument to this keyword. Only those entities where this regular expression does not match with

the string resulting from the `filter_attr` and `filter_fmt` keyword arguments will be included in the GELT. If the `filter_exp` keyword is present then both tests must pass for an entity to be included.

#### `centroid_attr`

The value of this keyword is the attribute list for creating the text that allows one to control on an individual basis what the centroid type is for each entity. The text that results from this is compared to the values associated with the keywords `median_value`, `user_value`, and `point_value` to determine the centroid type to use. This works in conjunction with the format string given as an argument to the `centroid_fmt` keyword to control how this text is generated. See [section 5.2](#) for an explanation of attribute lists and format strings.

#### `centroid_fmt`

The value of this keyword is the format string for creating the text that allows one to control on an individual basis what the centroid type is for each entity. The text that results from this is compared to the values associated with the keywords `median_value`, `user_value`, and `point_value` to determine the centroid type to use. This works in conjunction with the attribute list given as an argument to the `centroid_attr` keyword to control how this text is generated. See [section 5.2](#) for an explanation of attribute lists and format strings.

#### `median_value`

By default, the centroid type of entities generated from a shape file is an area weighted centroid. If the text resulting from the `centroid_attr` and `centroid_fmt` keyword arguments equals this, then this entity will use the point farthest from any boundary for the centroid that determines how far north, south, east, and west points are.

#### `user_value`

By default, the centroid type of entities generated from a shape file is an area weighted centroid. If the text resulting from the `centroid_attr` and `centroid_fmt` keyword arguments equals this, then this entity will use the a user supplied point for the centroid that determines how far north, south, east, and west points are.

#### `point_value`

By default, the centroid type of entities generated from a shape file is an area weighted centroid. If the text resulting from the `centroid_attr` and `centroid_fmt` keyword arguments equals this, then this entity will be considered to be a point entity.

#### `lat_attr`

Except for shape files, all input file types naturally come with a user-defined lat/lon point. If one needs a user-defined lat/lon point from a shape file, one needs to identify the attributes from which this information comes. The argument to this keyword must be the name of an attribute of type double that contains latitude values in degrees (north positive).

#### `lon_attr`

Except for shape files, all input file types naturally come with a user-defined lat/lon point. If one needs a user-defined lat/lon point from a shape file, one needs to identify the attributes from

which this information comes. The argument to this keyword must be the name of a attribute of type double that contains longitude values in degrees (east positive).

## **5.0) Text formatting and its application**

There are two ways that text is used in creating GELTs, and they both require very precise control of how text is formatted in order to function correctly. First, the text associated with each entity in the .id file is created in this way. Second, when an entity defined from one input file is modified somehow by the contents of another file, there must be a way to cross-reference each entity in the file with the specific entity in the other file from which the supplemental information is obtained. This is also done through text strings.

Section 5.1 describes how the various kinds of text associated with each entity work to perform this functionality. Section 5.2 describes exactly how to format a single text string.

### **5.1) Associated text and cross-referencing**

In [section 4.3](#), the reader will notice that there are four instances of paired keywords with `_attr` and `_fmt` endings. These correspond to four different types of strings that can be associated with each entity; output, unique, filter, and centroid text.

The output text associated with each entity is what eventually gets written into the .id file part of the GELT. To be useful, a regular file must have meaningful output text created, and thus both `output_attr` and `output_fmt` keywords must be supplied. If output text is defined for an override file, it only can be used for cross referencing. The concept of output text has no meaning for a special override file.

The unique text associated with each entity is used to determine when entities within an input file are to be considered redundant. If two entities in a file have the same unique text, then the information from the second one replaces the information from the first one. The exception is for polygons in a shape file, where the polygons from all file entities with that unique text are associated with that one logical entity. This all works exactly the same whether a file is a regular file or an override file. The ID string associated with each entity in a special override file is always considered to be unique text. For other files, if no `unique_attr` and `unique_fmt` keyword are supplied, the unique text defaults to the output text.

The filter text associated with each entity is used to eliminate certain entities based on the text associate with them. This feature is activated only if an argument has been supplied for one or both of the `filter_exp` and `filter_exp_not` keywords. If the filter text associated with an entity does not match the regular expression supplied with the `filter_exp` keyword, then that entity will be ignored. If the filter text associated with an entity matches the regular expression supplied with the `filter_exp_not` keyword, then that entity will be ignored. If text exists for both keywords, then the filter text must both match the `filter_exp` argument and not match the `filter_exp_not` argument to be included. This all works exactly the same whether a file is a regular file or an override file, but is not applicable to a special override file. If no `filter_attr` and `filter_fmt` keyword are supplied, the filter text defaults to the unique text.

The centroid text associated with each entity is used to control which centroid option is used with shape files. Other input file types have their own specific means of controlling this. The default centroid type is the area-weighted centroid. If the centroid text is the same as the value supplied with the `median_value` keyword, the the point furthest from the boundary will be used. If the centroid text is the same as the value supplied with the `user_value` keyword, the the user-supplied latitude and longitude will become the centroid. If the centroid text is the same as the value supplied with the `point_value` keyword, the entity will be defined by a single grid point at the user-supplied latitude and longitude.

If a regular file has one or more `override_with` keywords associated with it, then an attempt will be made to find a corresponding entity in that override file. If such a match is found, then certain characteristics of the entity in the regular file will be replaced with those in the corresponding entity from the override file. A match is found if either the output text, the unique text, or the filter text of an entity in the regular file matches any of those for an entity in the override file. By this we mean that a match could be found if an output text item from the regular file matched a filter text item from the override file. It does not necessarily have to be output to output, unique to unique, or filter to filter. This liberal matching approach is flexible, but also means one needs to be careful to avoid unintended matches.

When the override file is a shape file, the only characteristic that is acquired from the entity in the override file is a different geographic area for the entity. When the override file is not a shape file, the characteristics that will be transferred will be the centroid type, the user-defined location, and whether the entity should be an overlay entity (able to have assigned to it grid points already assigned to other entities). From a special override file, it is possible for an entity to be instructed not to get its geographic area from a shape file; this means it will either use a bcd file or be a point entity at the user-supplied location. From an ID file, the output text will be acquired.

## 5.2) Attribute lists and format strings

There are many instances where the values for two keywords ending in `_attr` and `_fmt` work in tandem to precisely format some kind of text. These represent an attribute list and a format string, respectively.

Here is an idealized model of what an attribute list looks like:

```
ATT_A|dAAA|dAAA ATT_B|dAAA ATT_C|dAAA|dAAA|dAAA
```

where the vertical bars are literal. Also, here is an idealized model of what a format string looks like:

```
%s some text | %f more text | %d
```

where the vertical bars and percent signs are literal. One should note that there is no necessary direct connection between the vertical bars in the attribute list and those in the format string. The ones in the attribute list are there because the vertical bar is the delimiter between attributes and each of their corresponding edit directives. The vertical bars in the format string are there because the text associated with each entity needs vertical bars to separate it logically into fields for the GELT software.

In the model attribute list, ATT\_A, ATT\_B, and ATT\_C are attributes of any type, of which there can be any number. Each attribute can have any number of edit directives attached to it, separated by vertical bars but no spaces. The `d' refers to the directive type, and the AAA refers to the directive argument. The edit directive is always a single character, but the argument can have any number of characters in it, or possibly be non-existent. The way the final output string is created is to interpret each attribute value with its corresponding format descriptor to create a piece of text, apply each edit directive in turn, then reinsert the resulting text into the format string to produce the final output string. The format descriptors in the format string are any legal C language format descriptors that end in `d', `f', or `s'. Examples would be `%d', `%s', `%f', `%10s', `%5.3f', `%4d', etc. The first format descriptor in the format string applies to the first attribute in the attribute list, the second format descriptor applies to the second attribute, and so on.

Here is a table of the edit directives and how their arguments are interpreted:

<b>directive character</b>	<b>function and argument interpretation</b>
#	Remove leading and trailing spaces, no argument used.
^	Make all upper case, no argument used.
v	Make all lower case, no argument used.
<	Prepend text of the argument.
>	Append text of the argument.
{	Argument is how many characters to remove from beginning of string.
}	Argument is how many characters to remove from end of string.
[	Argument is length to which to truncate string from the beginning.
]	Argument is length to which to truncate string from the end.
P	Parse string using first character of the argument as delimiter. Remainder of argument is which field to use in output: >0 means count from beginning, <0 means count from end.
p	Same as uppercase P except delimiter is a space or, in the case where the first character of the argument is a backslash, a vertical bar.

Here is an example. Suppose the text for your attribute list is

```
ZONE|#|<00|[3 STATE ZONE|#|<00|[3
```

and your format string is

```
%s | %sZ%s
```

Then, in the case where the value of the character ZONE attribute were "10 " and the value of the character STATE attribute were "WA", this would yield an output of "010 | WAZ010". Again, please note that the vertical bar in the format has no relationship to the vertical bars in the edit directives.

**Author: Jim Ramer**  
**Last update: 5 Jan 06**

## pasteUtil

pasteUtil is a standalone program meant to concatenate corresponding lines from text files together and write the resulting file to standard output; the usage is as follows:

```
pasteUtil entry entry entry ...
```

Each entry is either a file path or a static delimiter. It is assumed to be a file path if it does not begin with a `-'`. If it does begin with a `-', the delimiter does not include the `-'`.

Each line in the output file contains the data from each corresponding line in the input file, along with the static delimiters, in the order they appear on the command line. The delimiters are taken literally; no additional white space is added. To get white space, the user needs to quote the delimiters. The length in lines of the output file is the same as that input file with the greatest number of lines. Once one of the input files runs out of lines, it is assumed to contain enough empty lines after that to complete the operation.

---

**Author: Jim Ramer**  
**Last update: 6 Jun 98**

## processStyleInfo

processStyleInfo is a standalone program that produces the style entries for gridded data, using as input the contents of four tables: arrowStyle.rules, gridImageStyle.rules, contourStyle.rules, and iconStyle.rules. The files created are arrowStyle.txt, gridImageStyle.txt, contourStyle.txt, and iconStyle.txt. gridDepictKeys.txt is also modified to handle changes in display units required by the .rules files. For a complete description of how these grid tables work, one is directed to the file

localization/nationalData/styleRules.doc

The usage for processStyleInfo is as follows:

```
processStyleInfo {v} {x} {t} {c}
```

v (optional): Causes some diagnostics to be written to standard output.

x (optional): Completely expands out all key entries. Normally, many cross section style entries are abbreviated because they are so much like many other cross section style entries.

t (optional): Terse form, which is used in the localization. No titles in style entries.

c (optional): Causes keys to be compressed into base 32 format.

---

**Author: Jim Ramer**  
**Last update: 8 Jul 99**

## Range Azimuth Calculations

This program is a standalone program for doing range azimuth calculations. The results are directed to standard output. The units of vectors are kilometers, with azimuths in degrees from north.

The usage is as follows:

```
rangeAzimuth {i} {x} {v} lat lon arg3 arg4
```

i (optional): Output int integer format, otherwise floating point.

x (optional): Vectors are assumed to be cartesian, otherwise in range/azimuth.

v (optional): Input location and offset vector, outputting resultant location, otherwise input two locations, outputting vector.

lat: Latitude of starting location.

lon: Longitude of starting location.

arg3: Latitude of second location or range/x coordinate of vector.

arg4: Longitude of second location or azimuth/y coordinate of vector.

---

**Author: Jim Ramer**  
**Last update: 6 Jun 98**

## reformatTest

reformatTest is a standalone program that can be used to manage, test, and create netCDF files in the new format used for adaptable plan view plots. See [adaptivePlanViewPlotting](#) for more information.

One of the primary functions of this program is to allow the user to take netCDF data sets that were not created using the new NetcdfPointData class and convert them to that format.

This program will never cause previously existing output files to be deleted or have data removed from them. It will only create new time stamped files in a directory, add records to an existing file or overwrite existing data items in an existing record.

The usage for reformatTest is as follows:

```
reformatTest {s} {l} {a} {n} output input {delta {round {fcst} } }
```

s (optional): A literal `s', output data is considered static if present.

l (optional): A literal `l', output data is considered to be primarily identified by a lat/lon instead of a site ID if present.

a (optional): A literal `a', input is ASCII if present.

n (optional): A literal `n', program will issue notifications if present.

output: The directory (or file in the static case) where data will be written.

input: The path to the file from where the data will be read.

delta (optional): Times from input file will have this time offset added to them. 0 is the default.

round (optional): Times from input file will be rounded to this time offset. If 0 (the default) no rounding will occur.

fcst (optional): If supplied, an extra valid time will be supplied offset by this from the times in the input file.

The time offset values supplied in the `delta', `round' and `fcst' arguments are assumed to be in hours if the magnitude is less than 1800 and in seconds otherwise.

If the `s' option has been selected, the `output' argument needs to be the path to an actual file rather than a directory.

In either case, an actual netCDF file of the correct format needs to already exist. In the static case, the file itself must already exist. In the default case, a file named literally `template' must exist in the directory. The template or blank static files is generally created using the command `ncgen'.

The `input' argument is normally another netCDF file with roughly the same set of client variables, but not necessarily the `record management variables' specific to the new point data format. If the `a' flag is present, then the input file is an ASCII file the format of which follows.

If the `l' option has been selected, then this must be a data set where

records are primarily identified by lat/lon instead of an ID. If this is so, then the :idVariables attribute in the .cdl file for this data set will list the latitude and longitude variables.

Here is a idealized view of the contents of an ASCII input file for the reformatTest program. In this treatment the strings `//', `|', `id:' and `time:' are literal; everthing else is idealized.

```
// Comment
id: idtext | idtext ...
time: stamp | stamp ...
varName | dataItem | dataItem | dataItem ...
varName | dataItem | dataItem | dataItem ...
      :

id: idtext | idtext ...
time: timevalue | timevalue ...
varName | dataItem | dataItem | dataItem ...
varName | dataItem | dataItem | dataItem ...
      :

      :
      :
```

In this file, lines that are all comments, all white space, or blank are ignored. The records to write to are chosen by a combination of the id: and time: entries. After the id: string, there must be one vertical bar delimited string for each variable mentioned in the `idVariables' global attribute of the output data. For static data, the time: lines can be left out. Otherwise, the time: line must have one time value for each variable mentioned in the `timeVariables' global attribute. These time values need to be encoded just like an AWIPS file time stamp, such as 20000628\_1343 for 1343Z on June 28, 2000. Alternatively, the string `x' will be interpreted as a null time value, and a plain integer will interpreted as an offset to the immediately previous value (again less than 1800 hours, otherwise seconds). After an id: and time: entry, each line up until the next id: entry is assumed to be data for that record. The varName is the actual name of a client variable in the output netCDF file. The dataItem fields that follow are interpreted as values for each possible data item associated with that variable for the currently selected record. They will be interpreted as strings or numbers as appropriate for the type of the variable. For variables with just a record dimension (or a record dimension and a length dimension for character data), there will only be one value after the varName.

---

**Author: Jim Ramer**  
**Last update: 13 May 01**

## shp2bcd

This program is a standalone program for converting the information from ARCINFO shape files into AWIPS .bcd or .bcx files, which are the file formats that the workstation actually reads to draw map backgrounds.

The program allows the user two different methods by which to convert shape files into AWIPS cartographic data format. It also provides four different methods that allow the user to summarize the contents of a shape file in order to better use its contents. These are summarized here and explained in detail below:

- Convert the contents of a shape file to a .bcd file.
- Convert the contents of a shape file to a .bcx file.
- Write verbose summary of a shape file to standard output.
- Write table of selected attributes to standard output.
- Write bounding box of each shape to standard output.
- Write table of selected attributes to standard output using edit directives.

User note 1:

The second and the last usage mode make use of edit directives, which are summarized at the end of the file.

User note 2:

Path names to shape files should not contain the .shp, .shx, or .bdf file extension.

For the mode that converts the contents of a shape file to a .bcd file, the usage is as follows:

```
shp2bcd c shape_file bcd_file att_filter reg_exp
```

c: the letter `c'

shape\_file: path name to the input shape\_file.

bcd\_file: full name of the bcd file to write.

att\_filter (optional): name of character attribute to filter output with.

reg\_exp (optional): regular expression that value of `att\_filter' must match.

This section discusses the mode that converts the contents of a shape file to a .bcx file. For this usage mode, one should keep in mind that the workstation code that reads bcx files can only handle label lengths up to 30 characters. The usage for this mode is as follows:

```
shp2bcd x shape_file bcd_file att_names format att_filter reg_exp
```

x: the letter `x'

shape\_file: path name to the input shape\_file.

bcd\_file: full name of the bcx file to write.

att\_names: attribute names plus edit directives used to make labels.

format: format string used to make labels.

att\_filter (optional): name of character attribute to filter output with.  
reg\_exp (optional): regular expression that value of `att\_filter` must match.

For the mode that writes a verbose summary to standard output,  
the usage is as follows:

```
shp2bcd i shape_file
```

i: the letter `i`  
shape\_file: path name to the input shape\_file.

For the mode that writes a table of selected attributes to standard output,  
the usage is as follows:

```
shp2bcd a shape_file att_one att_two att_three ...
```

a: the letter `a`  
shape\_file: path name to the input shape\_file.  
att\_one: name of an attribute to tabulate the value of.  
att\_two (optional): name of another attribute to tabulate the value of.  
att\_three (optional): name of another attribute to tabulate the value of.

For the mode that writes the bounding box of each shape to standard output,  
the usage is as follows:

```
shp2bcd b shape_file att_one att_two att_three ...
```

b: the letter `b`  
shape\_file: path name to the input shape\_file.  
att\_one (optional): name of a character attribute to tabulate the value of.  
att\_two (optional): name of a character attribute to tabulate the value of.  
att\_three (optional): name of a character attribute to tabulate the value of.

For the mode that writes a table of selected attributes to standard output  
using edit directives, the usage is as follows:

```
shp2bcd e shape_file att_names format att_filter reg_exp
```

e: the letter `e`  
shape\_file: path name to the input shape\_file.  
att\_names: attribute names plus edit directives used to create output.  
format: format string used to create output.  
att\_filter (optional): name of character attribute to filter output with.  
reg\_exp (optional): regular expression that value of `att\_filter` must match.

Here is an explanation of how to mold an arbitrary list of attributes  
into the desired output string using edit directives.

When using the edit directives, the entire list of attributes to  
used for output needs to be in a single string, so in general this  
argument (`att\_names`) needs to be quoted, as does the `format`  
argument. One can replicate simple operations using the `e` mode with

no directives. For example, the following command using the simple tabulate operation

```
shp2bcd a shape_file ATT_A ATT_B ATT_C
```

would yield the same result as the following command using the edit directives mode

```
shp2bcd e shape_file "ATT_A ATT_B ATT_C" "%s %s %s"
```

on the assumption that each attribute listed was a character attribute. When using the `e' or `x' option, it is the user's responsibility to us a %s format descriptor for character attributes, a %f format descriptor for floating point data, and any of the various format descriptors that work with integer data for an integer attribute.

When actually using edit directives, the attribute list argument will look something like this

```
"ATT_A|dAAA|dAAA ATT_B|dAAA ATT_C|dAAA|dAAA|dAAA"
```

Each attribute can have any number of edit directives attached to it, separated by vertical bars but no spaces. The `d' refers to the directive type, and the AAA refers to the directive argument. The edit directive is always a single character, but the argument can have any number of characters in it, or possibly be non-existent. The way the final output string is created is to interpret each attribute value with its corresponding format descriptor to create a string, then apply each edit directive in turn, then reinsert the resulting string into the format to produce the output string.

Here is a table of the edit directives and how their arguments are interpreted:

directive character	function and argument interpretation
#	Remove leading and trailing spaces, no argument used.
^	Make all upper case, no argument used.
v	Make all lower case, no argument used.
<	Prepend text of the argument.
>	Append text of the argument.
{	Argument is how many characters to remove from beginning of string.
}	Argument is how many characters to remove from end of string.
[	Argument is length to truncate string to from the beginning.
]	Argument is length to truncate string to from the end
P	Parse string using first character of the argument as delimiter. Remainder of argument is which field to use in output; >0 means count from beginning, <0 means count from end.
p	Same as uppercase P except delimiter is a space or, in the case where the first character of the argument is a backslash, a vertical bar.

Here is an example. Suppose the text for your attribute list with edit directives were

```
"ZONE|#|<00|[3 STATE ZONE|#|<00|[3"
```

and your format string were

```
"%s | %sZ%s"
```

Then, in the case where the value of the character ZONE attribute were "10 " and the value of the character STATE attribute were "WA", this would yield an output of "010 | WAZ010". Note that the vertical bar in the format has no relationship to the vertical bars in the edit directives.

---

**Author: Jim Ramer**  
**Last update: 11 Jan 02**

# testDepictorTable

testDepictorTable is a standalone program that is a unit test of the DepictorTable class. A depictor table is used to optimize the remapping of points using two MapDepictor objects. Unlike other mapping tables that do all their mapping from and to integer indices, depictor tables remap from and to the floating point arbitrary cartesian coordinate systems of the two depictors. This is accomplished by breaking the area affected by the remapping into many tiles, each tile having two six-term polynomials for obtaining the output x and y coordinates. This allows this mapping to be accomplished using no higher math functions. For the case where the output depictor represents a radar data set, a mapping that occurs for points all near the radar will be done using a beta-plane to polar coordinate approximation, still optimized in a way that avoids any higher math functions.

The file containing the table will be written only if it does not already exist. Otherwise the file containing the table will be read in to create the internal representation of the depictor table. The filename of the table is `iiii__oooo.dpt.gz` where ``iiii'` is the name of the input map depictor file minus the `.sup` extension, ``oooo'` is the same for the output map depictor file, with the three underscores and the `.dpt.gz'` being literal. (If the ``n'` option below is included, the `.gz'` extension will not be present.)

The usage of testDepictorTable is as follows:

```
testDepictorTable in_map out_map {precision} {i} {t} {n}
```

`in_map` The name of the input geographic depictor.

`out_map` The name of the output geographic depictor.

`precision` If positive, desired maximum error in km. If negative, precision will be the size of the (optional) useful area of the mapping divided by this. Defaults to -3000.

`i` (optional) A literal flag that allows one to interactively test mapping individual points.

`t` (optional) A literal flag that allows one to do a timing test on the performance of the mapping versus a corresponding Xform.

`n` (optional) A literal flag that allows one to suppress the compression of the table upon output.

---

**Author: Jim Ramer**  
**Last update: 22 Apr 02**

## testFileNotify

testFileNotify is a standalone program meant primarily to generate a data notification based on the pathname to a file on the data disk. This must be a file that has a one on one correspondence to a key. Alternatively, it is possible to generate notifications for keys directly. Finally, it is also possible to run the program in a manner where arguments are taken from stdin.

For the mode that notifies files, the usage is as follows:

```
testFileNotify {-altDataKeys} pathName pathName pathName ...
```

The user may supply any number of `pathName' arguments, which must be the full path to the files to notify.

The optional -altDataKeys argument is a string that begins with a literal minus sign, followed by the name of a file that the program is supposed to read to get its data keys. For example, suppose the user knows that only radar data keys are going to be notified. By making the first argument -radarDataKeys.txt, the program will read in only radar data keys instead of all data keys, which greatly speeds up the program. If you want to know what the meaningful files for this argument are, look at the #include entries in \$FXA\_HOME/data/dataInfo.txt and nationalData/dataInfo.manual.

For the mode that notifies keys, the usage is as follows:

```
testFileNotify {-altDataKeys} k key stamp {f000} key stamp {f000} ...
```

The -altDataKeys argument is as before. The `k' is a literal k. The `key' argument is a data key to notify, and the `stamp' argument is a time encoded as a standard AWIPS file time stamp. In the optional `f000' argument, the leading f is literal, and the rest of the argument must be an integer, which corresponds to the forecast time to use in the notification. If that argument is less than 300, it is treated as as hours, otherwise as seconds. There may be any number of key/stamp entries, each one corresponding to one notification. Each key/stamp may or may not have an accompanying f000 argument. If omitted, it defaults to 0.

For the mode that takes arguments from stdin, the usage is as follows:

```
testFileNotify
```

Note that running the program with no arguments at all triggers this behavior. Each line received from stdin is assumed to be a set of arguments for the program as described above, not including the `testFileNotify' part. Only for the first set of arguments received in this manner is it meaningful to use the -altDataKeys argument.

---

**Author: Jim Ramer**  
**Last update: 14 Sep 00**

# Testing the Grid Key Server

testGridKeyServer is a standalone program that is used to do a unit test of the GridKeyServer object and to translate back and forth between keys and source/level/field descriptions for gridded data. It is also use to create depictor files for predefined lat/lon cross sections and to create volume browser menu files for predefined lat/lon cross sections and sources.

The usage is as follows:

- ```
testGridKeyServer {s} {p} {t} {v} {f} {g|G} {b} {q} {m} {k} {K} {a} {c} {d} {y}
```
- s (optional):  
List information about gridded data sources.
- p (optional):  
List information about gridded data planes.
- t (optional):  
List information about gridded data level types.
- v (optional):  
List information about virtual (derived) gridded data fields.
- f (optional):  
List information about raw gridded data fields.
- g or G (optional):  
Create depictor files for lat/lon cross section planes.
- b (optional):  
Create volume browser menu files for predefined lat/lon cross sections and sources.  
Source menus as OB8.3 and previous.
- q (optional):  
Create volume browser menu files for predefined lat/lon cross sections and sources.  
Source menus autogenerated with four categories instead of two as OB8.3 and previous.
- m (optional):  
Create volume browser menu files for predefined lat/lon cross sections and sources.  
Source menus generated from browserSourceMenuMaster.txt.
- k (optional):  
User will be prompted for grid data keys and they will be parsed into source/level/field components.
- K (optional):  
User will be prompted for source/level/field components and grid data keys will be provided.
- a (optional):  
Same as s, p, t, v, and f, and will allow one to test some lookups.
- c (optional):  
List CDL files, dimensions, depictor file, and topo file for active sources.
- d (optional):  
List directories and CDL files for active sources.
- y (optional):  
Build data key entries for active gridded data source directories (for purging).

User note 1:

Using the upper case version of flags b, q, and m will suppress writing out the browserSourceMenu\_n.txt files.

User note 2:

In order to get meaningful titles for cross section planes, one needs to have run the command ``testGridKeyServer G'`, which also depends on having access to depictor files for display scales. This has the most noticeable impact on using the p and B options of this program.

---

**Author: Jim Ramer**

**Last update: 30 April 08**

## testGridSliceWrapper

testGridSliceWrapper is a standalone program that was originally written as a unit test for the GridSliceWrapper class. It can also be used as a scripting interface into the AWIPS gridded data sets.

If invoked with no arguments, it invokes an interactive unit test, which is really only useful to developers. However, there are five other modes of operation which make it useful for script driven access to the gridded data.

For the mode that summarizes available sources, fields, and planes, the usage is as follows:

```
testGridSliceWrapper l
```

The ``l'` is a literal `l`. In this mode the IDs of all active sources are listed, then the IDs of all fields, then the IDs of all plan view planes, all one per line. The sources, fields, and planes are each separated by a blank line. The listing of field IDs includes a description if available. The fields listed are virtual fields, which means they do not necessarily have to represent a grid that exists in a netCDF file. They can represent stuff that is calculated on the fly.

For the mode that provides a data access key based on a source, field, and plane, the usage is as follows.

```
testGridSliceWrapper k sourceId fieldId planeId
```

The ``k'` is a literal `k`. The ``sourceId'`, ``fieldId'`, and ``planeId'` arguments are as provided by the ``l'` option. The key is printed to stdout. This key is needed for all access to the data. Of particular note is the `planeId` of ``3D'` which gives access to a three dimensional, multi level data set.

For the mode that provides inventory information, the usage is as follows:

```
testGridSliceWrapper key
```

The argument ``key'` is as provided by the ``k'` option. The available times are listed with initial times as a standard AWIPS file time stamp and the forecast times in both hours and seconds.

For the mode that reads gridded data, the usage is as follows:

```
testGridSliceWrapper key stamp fcst {file}
```

The argument ``key'` is as provided by the ``k'` option. The argument ``stamp'` is a standard AWIPS file time stamp. The argument ``fcst'` is forecast time. The forecast time is just an integer, assumed to be in hours

if  $\leq 300$ , otherwise in seconds. If the `file` argument is not provided, the grid(s) are retrieved and some summary information is printed to stderr. If the `file` argument is provided, the entire contents of the grid(s) retrieved are printed as ASCII to that file, a single dash (`-`) going to stdout. As the grids are written, each line is a row of the grid. The numbers are written in the same order as they are held in memory, so that the first row printed is actually at the bottom of the grid. If the data is vector data, both grids are output and the two grids are separated by a line with asterisks. For 3D data, a line with just the vertical coordinate value is printed before the grid. A vertical coordinate value of zero means the surface.

For the mode that reads gridded data and interpolates it to a specific point, the usage is as follows:

```
testGridSliceWrapper key stamp fcst file lat lon
```

The arguments `key`, `stamp`, `fcst`, and `file` are just as with the previous mode. The argument `lat` is latitude in degrees north and the argument `lon` is longitude in degrees east. Each grid retrieved as a result has its data interpolated to that point and the data is then output in ASCII to the file selected. Each level is output on one line; if there is only one level then just the data value(s) for that level are output. For 3D data, each line will start with the vertical coordinate value. Vector data is represented by two values.

---

**Author: Jim Ramer**  
**Last update: 06 Mar 01**

## test\_grhi\_remap

This routine is a standalone test routine for the grid remapping software in geolib. This routine is specifically meant to handle remaps with large numbers of grid points.

By default, both input and output gridded data is assumed to be in ascii format, one number per line. If an extra leading plus sign is placed on the input gridded data file name, then it is assumed that the input data file is an NGDC terrain data file. If an extra leading minus sign is used, then it is assumed that the input data file is a big endian version of such (i.e. one standard signed word per value).

If an extra leading plus or minus sign is placed on the output gridded data file name, then the output is scaled to be in the range of 1 to 254. If a plus, the data is output as a netCdf image data file. If a minus, then the output is just a stream of bytes (image). In the netCDF case, the file imageStyle.txt has a style info entry written to it. If the leading character on the output file name is an equals sign, then a file containing one standard signed word per value is written.

An output nx of zero will cause the program to clip rather than remap the data based on the area specified by the output depictor file. Program will print the output dimensions and write the depictor file clipArea.sup to describe the area of the clip.

This program has the following usage:

```
test_grhi_remap {n null} in_grid in_geofile in_nx in_ny \
                out_grid out_geofile out_nx out_ny {mult}
```

n {optional}: the letter 'n'; means specify the null value to use  
null {optional}: must be there if the 'n' is there, the null value to use  
in\_grid: name of file containing input grid data  
in\_geofile: name of geographic info file describing mapping of input grid  
in\_nx: number of gridpoints in the x direction in the input grid  
in\_ny: number of gridpoints in the y direction in the input grid  
out\_grid: name of file where output grid data is written  
out\_geofile: name of geographic info file describing mapping of output grid  
out\_nx: number of gridpoints in the x direction in the output grid  
out\_ny: number of gridpoints in the y direction in the output grid  
mult {optional}: what to scale style output by

Notes: A negative ny indicates grid indices increasing down.

---

**Author: Jim Ramer**  
**Last update: 29 Jul 98**

## testPlotDesign usage documentation

testPlotDesign is a standalone program that is a unit test for the ability to parse design files and use them to read data. (See [adaptivePlanViewPlotting.html](#) for more information on design files.) Its basic usage is as follows:

```
testPlotDesign designFile {dataKey} {station_id} {supfile} {maxdata}
```

The `designFile` argument need not include the full path...the software will use the standard AWIPS file locating logic to get the directory of the designFile. If the only argument supplied is the design file, then the designFile will be parsed and any errors reported.

If one additionally supplies a dataKey, the design file will be used to do an inventory on the data set to which the dataKey refers. If there is a leading minus sign (-) on the key, then the inventory will be tested but not output.

Supplying a station\_id will mean the inventory will be done for only that station. If one wants to supply arguments beyond the station ID but does not want to actually use a station ID, just supply " (two apostrophes) for that argument.

The supfile argument, if present, will trigger a test of data access using the given combination of design file, data key, and the geographic area covered by the sup file. If a blank (") argument is supplied, then the area covered will be the whole world. The sup file can be supplied without the directory and .sup extension.

When doing a data access test, the program will normally use the inventory to determine the times to test for, starting with the latest and going all the way to the earliest. One can limit the number of times to do a data access test for with the maxdata argument.

Normally performing a data access test will result in output from the test only if there are some sort of errors being reported or the diagnostics in the design file are turned on (see [adaptivePlanViewPlotting.html](#) section A1.1). However, if there is a scalar string item in the design file literally called "outputText", that text will be output to standard out.

---

**Author: Jim Ramer**  
**Last update: 07 May 04**

## textBufferTest

textBufferTest is a standalone program that can do a unit test of either the TextBuffer class or the InfoFileServer class. In either case, the program end up using the services of the InfoFileServer to find a file. If a file is found, some output is directed to standard out and the program returns and exit status of 0. If no file is found, will return an exit status of 1. The usage is as follows:

```
textBufferTest {-v} path_name
```

-v (optional): If string '-v' is present, program just verifies that the desired file exists and prints its full path name, otherwise uses the TextBuffer class to print the contents.  
path\_name: Path or partial path to give to the InfoFileServer.

User note:

For the unit test of the TextBuffer class, a zero length file is considered a failure condition.

---

**Author: Jim Ramer**  
**Last update: 6 Jun 98**

# va\_driver usage documentation

va\_driver is a standalone driver for the routines vis\_assign.c and va\_advanced.c. This program reads files which contain station locations, identifiers, and user preferences, producing a file which can be used by the workstation to display these stations or locations using a meaningful progressive disclosure strategy. User preference values have acquired the name 'goodness values', and the files containing these are referred to as 'goodness files'. All files that this program reads and writes are ASCII files where each line in the file refers to one station or location. This program always writes its output to the file va\_driver.out. Its usage is as follows:

```
va_driver goodness_file {=addtl_file} ... {primary_file}
      {s} {c} {l} {p} {a} {r} {f} {v} {V} {g} {m} {w weight} {d dist} {D}
```

goodness\_file:

file containing station locations, identifiers, and user preferences.

addtl\_file (optional):

additional file from which to get input stations/locations. Equals sign is not part of the file name, but is used to prevent it from being interpreted as a primary\_file.

primary\_file (optional):

file that contains a list of identifiers for stations that are important regardless of the preferences in the goodness file.

s (optional):

A literal flag which means the goodness file is a station file.

c (optional):

A literal flag which means the goodness file is a cities file.

l (optional):

A literal flag which means the goodness file is a location file.

p (optional):

Goodness values from 1 to 22222 are interpreted directly as progressive disclosure distances.

a (optional):

A literal flag which means call va\_advanced instead of vis\_assign to do the calculation; highly recommended.

r (optional):

If using va\_advanced, convert any progressive disclosure distances to goodness values. This allows one to catenate existing progressive disclosure files and reprocess them to make an internally-consistent set of PD values.

f (optional):

A literal flag which means, for location files, consider the aspect ratio of the footprint of what is to be plotted in doing the calculations.

v (optional):

Print to stdout the character s, c, or l based on the type of file being processed.

V (optional):

Print to stdout the character s, c, or l based on the type of file being processed, then immediately exit without doing a calculation.

g (optional):

The goodness values finally used in the computation of progressive disclosure are increased each time a station repeats in the input data.

m (optional):

Allow redundantly-named entries to be processed...otherwise will use only last occurrence of redundantly-named entries.

w (optional):

A literal flag which means invoke a feature that allows one to balance user preferences versus spatial uniformity.

weight (optional):

A number from 0.0 to 1.0, where 0 means consider only user preferences, 1 means consider only spatial uniformity. 0 is the default.

d (optional):

A literal flag which means invoke a feature that allows one to consider separation in determining station uniqueness.

dist (optional):

If d flag is present, stations with the same identifier but further apart than this will still be considered unique. Defaults to 100km.

D (optional):

A literal flag which means separate raw Cities file records into unique and duplicates. Output as cities.unique and cities.duplicate.

User notes:

This program operates in three main modes controlled by the 's', 'c' and 'l' flags. If the 's' flag is present, the input file is a station goodness file and the output file is a station plot info (.spi) file. If the 'l' flag is present, the input file is a location goodness file and the output file is a location plot info (.lpi) file. If the 'c' flag is present, the input file is a cities file and the output file is also a .lpi file. If none of these flags is present, the program will try to determine the format. By convention, both station and location goodness files have a '.goodness' extension and primary files have a '.primary'. The extensions of '.spi' for a station plot information file and '.lpi' for a location plot information files are mandatory. If additional input files are given as arguments, they must all have the same format as the first input file. Additional input files will always interpret floating point goodness numbers as goodness values rather than progressive disclosure distances. There is currently only one cities file permanently in existence, called CitiesInfo.txt. As mentioned above, all files that this program reads and writes are ASCII files where each line in the file refers to one station or location. The formats of each line in these files are as follows:

### **station goodness file:**

number name latitude longitude elevation goodness accessId

'number' is an integer corresponding to the station number of the station. 'name' is what primarily identifies each station to users. 'latitude' and 'longitude' are decimal degrees with east and north positive. 'elevation' is an integer that specifies station height in meters. Normally, 'goodness' is an arbitrary integer that says how desirable a given station is to plot in lieu of another when two stations are too close to avoid an overlap. The larger the number the more likely a station will show up on a plot. However, if 'goodness' is a floating point number (contains a decimal point) then that value will be used directly as a progressive disclosure value, which is the distance in kilometers to the nearest other station that is at least as visible. 'accessId' is an optional field that contains the string by which a station is identified for the purpose of data

access. If not present this will default to `name`. The `name` field is currently limited to 11 characters, and the `accessId` field is currently limited to 23 characters.

Historically the `number` was the WMO station number or some analogue. All access of station data in AWIPS is now according to the ASCII station ID, and so the station number is an obsolete concept. Thus, this field has come to be used for the station-specific data key, where such a key exists. Thus, for most data sources, this is just always zero.

Additionally, if a line is present with only two real numbers, then those numbers are used as the minimum and maximum allowable progressive disclosure distances. If the same station name occurs more than once, only the last occurrence will be used.

### **location goodness file:**

latitude longitude goodness name

All fields are just as in the station goodness file, except that the `name` field can have any number of spaces in it. The `name` field is currently limited to 39 characters. If a vertical bar occurs in the `name` field, the vertical bar and everything after it is ignored for plotting purposes, but these characters can still mark a `name` as being unique.

Normally, if the same location name occurs more than once, only the last occurrence will be used. However, if a line is present in the file that has just a single `m` on it, multiple occurrences of a location name will be allowed. Just as with the station goodness files, if a line is present with only two real numbers, then those numbers are used as the minimum and maximum allowable progressive disclosure distances.

### **cities file:**

latitude longitude goodness state name|n?

The fields latitude, longitude, and goodness are as before. The field `state` normally just contains a two-character postal abbreviation. Optionally, an additional character can follow the postal abbreviation in the `state` field that allows two cities with the same name in the same state to nonetheless be identified as unique. Unlike other fields, the `name` field can have any number of spaces in it. If two or more entries occur that have exactly the same `name` and `state` fields, only the last entry is used. If the `|n?` is present, then this city is important to the warning function. If n is 1 it is a major city, if n is 3 it is a minor location, and 2 is a location of average importance. The `?`, if present, is the character `=`, `+`, or `~`, and means that this city is too large in area to be considered a point. `+` means use an area weighted centroid, `~` means use the point furthest from any border as the centroid, and `=` means use the supplied point as the centroid. The centroid is a point used to calculate what is north, south, east, or west in the city.

### **station plot info (.spi) file:**

number name latitude longitude elevation distance accessId

The `number`, `name`, `latitude`, `longitude`, `elevation`, and `accessId` items are exactly as in the station goodness files. `distance` is the parameter that is used for controlling progressive disclosure. It is the distance in kilometers to the nearest station that is at least as likely to be plotted.

**location plot info (.lpi) file:**

latitude longitude distance name

The `latitude`, `longitude`, and `name` items are exactly as in the location goodness files. `distance` is the parameter that is used for controlling progressive disclosure. It is the distance in kilometers to the nearest station that is at least as likely to be plotted.

**"primary" file**

A .primary file contains a list of station or location names that are meant to be considered as important regardless of the goodness values present in the .goodness file. The first station in the list is considered most important, the second next, and so on. If a station or location appears twice, the first occurrence is what is used.

---

**Author: Jim Ramer**  
**Last update: 31 Aug 05**