April 6, 2011

**Setting up the bundle converter**

In order to use this prototype, stage all the files in the same directory, in an environment that has access to an old D-2D localization data set. One should also have all the standard old D-2D environment variables set. This needs to be run on a machine with Redhat 5.

If one needs to run this on a machine that does not have a working old D-2D environment, two files are provided that support implementing just enough of this enviroment to allow the bundle conversion to function. The script grabFromD-2D.csh is meant to be run on a machine with an old D-2D environment; it tars up just enough of that environment to support bundle conversion. The resulting tar (convBundStage.tgz) is meant to be untarred into the directory /awips/fxa on whatever machine one wants to perform the bundle conversion. This directory should first be created if it does not exist. Once these files have been untarred, sourcing the file convBund.cshsrc will set up the environment variables needed to allow the program convertBundles to function. Be aware that convBund.cshsrc is meant to be sourced, not run as a command, and it can only be successfully sourced from within a C shell. The program convertBundles does the work; the rest of the files are supporting data. The program convertBundles will try to locate a version of the file LevelMappingFile.xml. By default it will try to find it somewhere under /usr/local/viz/cave/plugins/. Alternatively, it will look under $CAVE\_ROOT/, or one could stage a version of LevelMappingFile.xml in the same directory as convertBundles is in. You should be able to run the program convertBundles from anywhere, as long as it is staged in the same directory as the supporting data.

**Running and testing the bundle converter.**

There are two primary usages for the bundle converter program:

convertBundles netcdfPath

convertBundles netcdfPath xmlPath ...

In these idealized usage examples, netcdfPath is the full path to any old style D-2D netcdf bundle file. The first usage writes the text of a Cave xml bundle to stdout. For the second usage, xmlPath is the full path to any file to write the text of a Cave xml bundle to; there can be any number of netcdfPath - xmlPath pairs. Optionally, one can supply a leading literal 'a' argument. If a useable 'overlay' entry does not already exist for a given depict key in BundleConversionInfo.txt, the 'a' argument will cause the file overlay-kkkk.xml (where kkkk is the depict key for the overlay) to automatically get used if it exists. One does not necessarily have to be in the same directory as the convertBundles program in order to successfully run it. However, the convertBundles program and all its associated metadata do need to all be staged together in the same directory. Be advised that there are no smarts in the convertBundles program as far as organizing bundle files in the file system. All this knows how to do is convert old D-2D netcdf bundle files to A-II xml bundles. You have to tell it specifically what files to use as input and output. In a legacy system, netcdf bundles typically reside at a path like:

/data/fxa/userPrefs/$USER/procedureName/FXA.\*

For now, probably the best way to test individual converted bundles is to dump them in your home directory on your Cave host. Then you can load them using the file chooser dialog on Cave located at:

Cave -> Load/Save Displays -> Load Displays...

With this version of the converter is a new script called convertProc.csh. In the legacy D-2D, what we call a procedure is implemented by a directory that contains netcdf bundle files plus an index file. The analogous data structure in A-II is totally self contained in a single xml file. The script convertProc.csh takes the path of one procedure directory as an argument and tries to produce an A-II procedure xml file, written to the current working directory; the name of the output file will be reported by the script to stdout. Many procedure directories may have spaces in the directory name, and in that case one may need to quote the procedure directory path. This script uses the literal 'a' option with the convertBundles program, so it will automatically try to pick up files named overlay-kkkk.xml (where kkkk is the depict key for the overlay) if they exist in the converter directory. Currently, one location that an A-II procedure xml file can be placed in is:

~/caveData/etc/user/$USER/procedures/

In the current implementation, this should be able to convert any VB overlay that A-II actually implements. It will also convert just about any radar image except except for mosaics with composite reflectivity, and it will probably convert most radar graphics. Satellite images, redbook graphics, raob skew-Ts and profiler time-heights should all be fully implemented. Nearly all multi-level plan view plots (including profiler and Raob) are in place, plus one metar plot, the standard hourly plot. Lightning, severe weather plots, local warnings, and SPC watch displays have been implemented as well. Besides map backgrounds that are scale defaults, only point map backgrounds have been implemented.

Be advised that while most overlays will be converted, there are many overlays that will not. If there is a particular overlay that the converter is not yet designed to handle, but the user considers necessary, the user is referred to the next section, entitiled “Customizing the bundle converter."

The bundle converter has gotten mature enough such that if the documentation says a particular type of overlay should be implemented, but it fails to get translated, it should be reported as a bug in the bundle converter. One caveat to this is that there are still some things that D-2D can load but Cave cannot. Another caveat is that there are certain things that can be loaded on Cave through the menus, but if you create a bundle in Cave and then try to reload using the Cave generated bundle, it will fail. If either of these things is true, this should be reported as a Cave bug rather than as a bug in the bundle converter.

As this is written (April 29, 2011) bugs in the bundle converter should be reported to Ashley Kells (Ashley.Kells@noaa.gov) and/or James Ramer (james.e.ramer@noaa.gov).

**Customizing the bundle converter.**

For any radar, VB, or satellite image displays that cannot be successfully be encoded into an xml bundle, GSD will take responsibility for seeing to it that these are addressed. However, for other types of displays, especially those where an overlay xml file is supplied for one specific depict key, it is hoped that people outside of GSD can be leveraged to fill out the set of displays that be successfully handled. To that end, the script locateKey.csh has been provided. To successfully craft an entry in BundleConversionInfo.txt that can be associated with an overlay, at a minimum one needs either the depict key or the depictable type. If one runs locateKey.csh with a legend or legend fragment as an argument, it will attempt to lookup that overlay in the legacy D-2D tables and print out the information needed for entries in BundleConversionInfo.txt.

The easiest way to implement a specific xml file for an overlay is to first load the product of interest on Cave, then unload everything except the overlay in question, including the map backgrounds (unless it is a map background one is trying to implement). Then save the bundle to a file using the chooser dialog on Cave located at:

Cave -> Load/Save Displays -> Save Editor Displays...

Your overlay specific xml is then everything in that file from the first line with <resource> to the last line with </resource>. Place a file with the data for the overlay into the converter directory, and add an entry to BundleConversionInfo.txt that looks like this:

overlay | kkkk | myOverlaySpecific.xml

where 'overlay' is literal and 'kkkk' is the depict key, most easily looked up using the locateKey.csh script described previously.

More information on the format of overlay templates.

In the set of xml files supplied with the converter, once sees a number of instances where text between two at signs (@) occurs. The first category of these are symbols that result in a direct text replacement and are specifically tailored for a particular display type. In general, these are

fully implemented and users should not need to place them in their own overlay specific xml. Here is the current set of those symbols:

@VRTSCL\_NAME@ @BOTTOM@ @TOP@

@LINE\_ID@ @LINE\_STR@ @POINT\_LAT@ @POINT\_LON@

@VB\_SOURCE@ @VB\_LEGEND@ @VB\_FIELD@

@VB\_LEVEL\_ONE@ @VB\_LEVEL\_TYPE@ @VB\_LEVEL\_TWO@

@DB\_NAME@ @STATION\_INFO@

@EXTRA\_XML@ @LAT\_MIN@ @LAT\_MAX@ @LON\_MIN@ @LON\_MAX@

@RADAR\_ID@ @RADAR\_MSG\_CODE@ @CODE\_LIST@ @TILT\_MIN@ @TILT\_MAX@

@SAT\_CHANNEL@ @SAT\_SECTOR@ @SAT\_ENTITY@

@WMO\_ID@ @NNN\_LIST@

@ENV\_MIN\_X@ @ENV\_MAX\_X@ @ENV\_MIN\_Y@ @ENV\_MAX\_Y@ @MAX\_RANGE\_X@

@PROJ\_NAME@ @PROJ\_LAT@ @PROJ\_LON@

Another category includes a couple of symbols that result in a very specific substitution, but that do have general use. The first is the symbol @OVERLAY\_LEGEND@, which as its name suggests gets replaced with the legend of the overlay. The second is @COLOR\_MAP\_INFO@, which must be included in any image overlay, and needs to appear on a line by itself in the following context:

<capability xsi:type="colorMapCapability">

@COLOR\_MAP\_INFO@

</capability>

Most of the rest of the symbols discussed here have arguments. A symbol with arguments always appears on a line by itself and generally looks like this:

@NAME\_OF\_SYMBOL,argument1,argument2@

In certain cases a symbol with allow an alternate delimeter besides a comma, but the same delimiter must be used for the entire text of the symbol. Where allowed, here is the set of available alternate delimiters:

" \ ' - . / : ; ` |

Also, in certain cases the text of a symbol with arguments is allowed to span multiple lines as long as the last non-space character on the line is a delimiter or the terminating at sign. A multi-line symbol could look like this:

@NAME\_OF\_SYMBOL, argument1, argument2, argument3,argument4,argument5,

argument6, argument7@

For all symbols with arguments, spaces adjacent to the delimeters and trailing at sign do not become part of the arguments. In all cases, the text of the symbol itself is stripped once it has been used to direct some modificaton of the text, and in most cases it does not matter where in the text the symbol appears.

The symbol with the name DATA\_KEY\_TEXT is used to pull text into an overlay template that is specific to individual data keys. It has one argument that is the name of the file from which to pull in text to represent each data key. For the @DATA\_KEY\_TEXT symbol, it does matter where in the text it appears; this controls the point in the main xml template where the datakey text is inserted. A file pulled in this way can have its own nested occurence of @DATA\_KEY\_TEXT in some circumstances. Sometimes this symbol occurs with no file name argument, in which case there must be an entry in BundleConversionInfo.txt with the 'datakey' type active for the overlay template. For the most part, the needed occurences of @DATA\_KEY\_TEXT are fully implemented and users should not need to place them in their own overlay specific xml. For many instances of plan view plots in the legacy D-2D, the 13th vertical bar delimited field in the depict key contains the so-called 'extra info', which is a series of comma delimited strings. In the legacy D-2D, the extra info performs a symbol value substitution in the associated design file, and this is used in much the same way in the bundle converter. For depict key 2500, which is the 500MB raob plot, the extra info contains levelMB and 500. In the corresponding xml overlay template, this results in @levelMB@ being replaced with 500. This is the only situation where additional symbols can be defined dynamically based on the nature of the overlay being converted, and symbols so defined never have arguments. Furthermore, conversions based on the extra info can be made more flexible using the @TRANSLATE symbol. Here is an example of an instance of using the @TRANSLATE symbol:

@TRANSLATE,levelIdx,0,0.5km,1,750m,2,1km,3,1250m,4,1.5km@

What this means is that where the extra info by itself would cause @levelIdx@ to be replaced by 0, now it will be replaced by 0.5km, where it would be replaced by 1, now it will be replaced by 750m, etc. The text of the @TRANSLATE symbol can span multiple lines, but only uses the default comma delimeter because the extra info text itself is always comma delimeted. As more of the plan view plots get implemented, users may find themselves creating additional xml templates that interact with the extra info in these ways. Finally, there are three additional ways in which symbols can cause the xml to be modified, and these are not limited in any way to a particular type of display. The first, and simplest, is the symbol @REPLACE, which looks like this:

@REPLACE,textToReplace,textToInsert,textToReplace,textToInsert@

There can any number of replace-insert pairs. The @REPLACE symbol allows alternate delimeters and can span multiple lines. The next general means of symbol driven text modification is the @REPLACE\_FOR\_KEY symbol. It has three usages:

@REPLACE\_FOR\_KEY,DEPICT,depictkey,textToReplace,textToInsert@

@REPLACE\_FOR\_KEY,DATA,datakey,textToReplace,textToInsert@

@REPLACE\_FOR\_KEY,key,textToReplace,textToInsert@

The first usage performs the text replacement only if the supplied keyis the same as the depict key for the D-2D overlay. The second usage performs the text replacement only if the supplied key is the same as one of the data keys for the D-2D overlay. The third usage can match either depict or data keys. In all cases, there can be any number of key-replace-insert triplets. The @REPLACE\_FOR\_KEY symbol allows alternate delimeters and can span multiple lines.

The final general means of symbol driven text modification actually involves two symbols working in concert, and in this case it does matter where in the text the symbols are placed. The two symbols are @REMOVE\_FOR\_KEY and @END\_REMOVE@. The @REMOVE\_FOR\_KEY symbol is placed just before the first line of some text one might want to remove from the xml and @END\_REMOVE @ is placed on a line just after the last line of some text one might want removed. The @END\_REMOVE@ symbol has no arguments, and the @REMOVE\_FOR\_KEY symbol has any number of arguments that are keys. If any of the keys match, the text between the two symbols gets removed. As with the REPLACE\_FOR\_KEY symbol, one can include the optional leading DEPICT or DATA arguments. The @REMOVE\_FOR\_KEY symbol allows alternate delimeters and can span multiple lines.