# Table of Contents

***Release***

**Mar 24, 2017**

# Table of Contents

**Note:** This guide is for Focal Points who are already familiar with the material in the Hazard Services User's Guide. It covers basic, required configuration as well as advanced optional customizations.

**Warning:** This is a work in progress. See Action Items for additional items to be covered in this guide.

CHAPTER 1

# Getting Started

## Define "possibleSites" and "visibleSites" (Required)

> **Warning:** Without this patch, drawing a polygon and then choosing a hazard type in the Hazard Information Dialog **MAY** result in the polygon completely disappearing.

**Action:** Modify *common_static/site/LLL/HazardServices/startUpConfig/StartUpConfig.py* to contain the following:

```
StartUpConfig = {
  "possibleSites": ["LLL"],
  "visibleSites": ["LLL"]
}
```

## Define backup sites if needed

This step is only necessary if there is a desire to interoperate with service backup. Of course, any site using Hazard Services to fulfill operational responsibilities will no doubt need to take this step. To do this, create a file at the path common_static/site/LLL/HazardServices/settings/backupSites.xml with the following contents:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<backupSites>
        <sites>BBX</sites>
        <sites>BBY</sites>
        <sites>BBZ</sites>
</backupSites>
```

In this idealized example, BBX, BBY, and BBZ are not literal, but rather represent the site id's of the neighboring sites you want service backup functionality for. In order for Hazard Services to properly function for these backup sites, each backup site will need its own site override of StartUpConfig.py defining **"possibleSites"** and **"visibleSites"** (see

Section 1). Furthermore, after entering service backup it is desirable to be able to switch back to your primary site. To enable this, a site override of backupSites.xml is needed for each of your backup site ids containing the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<backupSites>
        <sites>LLL</sites>
</backupSites>
```

## Synchronize hazard inclusion percentages

This impacts interoperability, which is the ability for hazards issued in warnGen to also be handled in Hazard Services and vice versa. If all the inclusion percentages for warnGen are the base defaults, then there is no need to take this step. Run the simple shell script that follows on a px or dx; if it produces no output, then in all likelihood you have no overrides for include percentages that impact IOC products for Hazard Services:

```
#!/bin/csh -f
set LLL = `cat /awips2/edex/bin/setup.env | grep AW_SITE_IDENTIFIER | \
           head -n 1 | cut '-d=' -f2 | tr -d ' '`
set cmnRoot = /awips2/edex/data/utility/common_static
if ( -e $cmnRoot/site/$LLL/warngen/geospatialConfig_COUNTY.xml ) then
    diff $cmnRoot/site/$LLL/warngen/geospatialConfig_COUNTY.xml \
         $cmnRoot/base/warngen/geospatialConfig_COUNTY.xml | \
         grep inclusionPercent
endif
if ( -e $cmnRoot/site/$LLL/warngen/geospatialConfig_ZONE.xml ) then
    diff $cmnRoot/site/$LLL/warngen/geospatialConfig_ZONE.xml \
         $cmnRoot/base/warngen/geospatialConfig_ZONE.xml | \
         grep inclusionPercent
endif
set templateOverrides = \
   `( cd $cmnRoot/site/$LLL/warngen ; find . -name '*xml' ) | \
    cut -c3-99 | grep -i flood | grep -v geospatialConfig`
foreach one ( $templateOverrides )
    grep inclusionPercent $cmnRoot/site/$LLL/warngen/$one
end
#
```

**download** the above example

If there are overrides of inclusion percentages for warnGen, one needs to override the inclusion percentages for Hazard Services to match them. This is done by producing an override file for HazardTypes.py. The paths, respectively, to the base and site override version of this file are:

```
common_static/base/HazardServices/hazardTypes/HazardTypes.py
common_static/site/LLL/HazardServices/hazardTypes/HazardTypes.py
```

This file is subject to incremental override, and so the site override file need only contain the new inclusion percentages for any impacted hazard types. Suppose for example that the only change needed was to adjust the inclusion percentage for Convective FFWs to 15 percent. Then the entire contents of the site override of HazardTypes.py would be:

```
HazardTypes = {
   'FF.W.Convective' : {'inclusionFraction': 15}
}
```

Changes in geospatialConfig_COUNTY.xml would potentially impact every hazard type where you have 'ugcType': 'county', and changes in geospatialConfig_ZONE.xml could potentially affect all those where you have 'ugcType':

'zone'. If the inclusion percentage is overridden in an individual product template, then one needs to identify the associated hazard in HazardTypes.py and provide the appropriate override value.

For a standard production installation of Cave including the Hazard Services software, the Hazard Services application really won't do much that is very useful. In order for Hazard Services to function correctly, there is a bare minimum amount of manual site configuration needed.

This section will focus on what is absolutely needed to get Hazard Services to the point where it will issue SOMETHING. We will also document some customizations that prevent certain incorrect behaviors from manifesting. Later sections will discuss how to customize Hazard Services such that the look and feel of the products issued conform to what is specifically desired for your WFO.

Here we will focus on which content will need to change in which files. We will use LLL to stand for the primary WFO id of your site. Where file paths are mentioned that start with common_static/ or cave_static/, it is implied that these are subdirectories of /awips2/edex/data/utility/. For changes to configuration files in EDEX, it will be left to the reader to decide whether to implement the changes in the Localization perspective or by changing the files in the file system and then bouncing EDEX. Changing files in the Localization perspective is always preferred, but every once in a while problems will manifest with how userRoles work, and there will be no way around it other than to change a file in the file system and then restart EDEX. On an lx, one can inspect the contents of these same files under ~/caveData/common/ (for common_static/) or under ~/caveData/etc/ (for cave_static/), **but under no circumstances should one directly modify files under ~/caveData/common/ or ~/caveData/etc/.** Where these instructions request adding or changing a file that for some reason your site has already added or changed, then the changes will need to be merged. Some of the changes mentioned here may need to be updated later in order to implement a robust rather than bare bones functionality.

CHAPTER 2

# Customization

Hazard Services is delivered with a set of baseline hazard generation functionality. However, it is clear that "one size" does not fit all national sites spanning diverse meteorological regimes and topographies. Therefore, Hazard Services is highly configurable and this section will help Focal Points to customize the system to address their local requirements.

We describe the various recommenders, product generation files, and configuration files showing how they are used by the system. We include use cases and instructions for customization.

## Localization

Hazard Services needs to be localized to your site. This document covers Hazard Services-specific steps needed and assumes that more general localization steps such as porting over a Hydro Database, setting up FFMP, and localizing EDEX have been taken care of.

The Localization Perspective in CAVE provides access to the user-configurable files for various applications. There is a directory for Hazard Services which contains configuration files, recommenders, and product generators. Each file has a BASE version which can be viewed in the Localization editor by clicking on it. A copy of this file can be made to produce a SITE, Desk, Workstation, or User version. If these versions exist, they are used instead of the BASE version in a hierarchical manner. For example, if a SITE version exists, it is used instead of the BASE. If a User version exists, it is used instead of the SITE or BASE. Overrides can be partial in the sense that the override file contains only what is to be changed from the base file. In this way, specific customizations can be made.

### Overrides

In Hazard Services, there are various kinds of Overrides to files:

- **Incremental Override:** Some files consist of simple Python dictionaries or lists and are overridden by a process we call "incremental" override. Examples of these files are the Settings files (e.g. Hydrology_All.py) and HazardTypes.py. Although you will see examples of overriding files, a complete set of documentation for "incremental override" of Hazard Services configuration files can be found in the Incremental Override for Configuration Data document.

- **Class-based Override:** Other files contain Python classes and their overrides are "class-based". A "skeleton" class is defined in the override file and the user adds selected Python methods from the class to "override" or add to the original methods. Overriding of these files is similar to the GFE and is covered in that training.
- **XML Override:** Some configuration files (e.g. Alerts) are "xml" format and are overridden as other "xml" files in AWIPS 2.
- **Python Method Override:** Some files consist of only Python methods e.g. Megawidget Side Effects. To override these files, you must copy the entire method into your override file.

There may be times when the more conventional non-incremental override behavior is desired even though the file is subject to incremental override. This is very simple to achieve with a single extra entry placed at the very beginning of the override file content; the form of this entry depends on whether the file is a dictionary or list at the top level. For a dictionary this extra entry is **"_override_replace_" : True**, ; for a list it is **"_override_replace_"**, . Then the content that follows can be a complete copy of the base, modified in whatever way is desired.

The Localization Perspective under the Hazard Services tab (or in directories under ...utility/common_static/base/hazardServices/) contains the baseline and localization files for Hazard Services. You will find the following tabs in alphabetical order. They are listed here in logical groupings:

- **Configuration of Graphical User Interface**
  - Startup Config (incremental override)
  - Alerts (xml override)
  - Settings (incremental override)
- **Hazard Types and metadata**
  - Hazard Types (incremental override)
  - TODO: Hazard Type Color Table ??
  - Hazard Categories (incremental override)
  - Hazard metadata (class-based override)
  - Megawidget Side Effects (python method override)
- **Recommenders and Product Generation**
  - Recommenders (class-based override)
  - Product Generator Table (incremental override)
  - Product Generators (class-based override)
  - Product Formats (class-based override)
  - Utilities (class-based override)

## Megawidgets

Hazard Services supports a set of user-defined graphical user interface components, called Megawidgets which are used throughout Hazard Services customization. Examples will be seen throughout this document and complete documentation can be found in the Megawidget Section.

## Required Steps

1. **StartUpConfig.py:** Override StartUpConfig.py required fields as designated in the baseline file.

2. **Site-specific geometries:** Ingest shapefiles for Dam Inundation, Riverpoint Inundation, Burnscar areas. See detailed instructions in the next section.

### Optional Steps

- **Startup Configuration:** Override non-required items in StartUpConfig.py which has various options that you may want to change.
- **Settings:** Create Site-Specific Settings. It may be useful for your site to have site-specific Settings to aid in the Forecast Process. For example, you may want a Setting that views only Warnings and Advisories, filtering out Watches. Or you may want a Setting to view only issued and ending hazards.
- **General Configuration:** Baseline versions of the Recommenders, Product Generators, Hazard MetaData (appearing in the Hazard Information Dialog), Hazard Types and Categories will work for your site "out-of-the-box." However, as you work with the system, you will want to appropriately localize. This document will give you the information you need to do so.

---

**Note:** Configuration of Hazard Services involves Python configuration files. If you are not already familiar with Python and Class-based programming there are various tutorials that will give you the background needed:

- Codeacademy –Gives problems and checks them for you.
- Google tutorial –Has videos which is also very helpful.
- Official Python Website
- Learn Python
- Tutorials Point

---

## StartUp Configuration

The StartUpConfig.py file contains various items listed below. More detailed information can be found in the documentation within that file.

- Dissemination order for products
- Recommender to use when creating hazards from gage points
- Console Settings and Time Line navigation options
- Hazard Information Dialog layout and tab options
- Specific default Alerts

## Hazard Configuration

The baseline VTEC hazard types, categories, and metadata are set up in the files discussed in this section. All of these files can be overridden to adjust modifiable attributes of existing hazards or to add new hazard types.

## Hazard Types

The Hazard Types are stored in a localization file (HazardTypes.py) identifying all the hazards and basic information about each. (This is similar to the VTECTable in legacy operations.) It's stored as a Python dictionary of dictionaries:

```
dict = {phen:{TO.W:field}, sig{FF.W.Convective:field}, sub-type{FF.W.
↪NonConvective:field}}
# sub-type is optional
# An example NonConvective Field is a dam failure
# Please see the *HazardTypes.py* file for descriptions of the fields
# Example headline: 'FLASH FLOOD WARNING'
```

**Note:** The hazard Types file contains almost 100 hazard types and is quite large. Using incremental override to add a new hazard type, is easy. Just modify the SITE level file like the example below.

```
HazardTypes = {
  'HY.S' : {'phen': 'HY',
            'sig': 'S',
            'headline': 'HYDROLOGIC STATEMENT',
   }
}
```

Also, since Hazard Services will eventually implement a National Hazard Database, there is the possibility that with user-defined hazard types there could be duplicate hazard types with different meanings which could lead to confusion for Forecasters viewing hazard information from other sites. This problem needs to be addressed, perhaps as a National registry for hazard types.

## Hazard Categories

Hazard Types are designated user-configurable Hazard Categories which are defined in another localization file containing a dictionaries of Python List with entries. The dictionary is a mapping from a category name to a list of hazard types (and possibly subtypes).

The Hazard Categories make it easier to select the desired Hazard Type in the Settings dialog and the Hazard Information Dialog. Incremental override could be used on this file as well, to add to or modify the BASE configuration. For example, to move convective watches into a new category called, literally, "Convective Watches", the SITE override file might contain either of the following:

```
HazardCategories =
        {
        "Convective": ["_override_remove_list_", ("SV","A"), ("TO","A")],
        "Convective Watches": [ ("SV","A"), ("TO","A") ]
        }
HazardCategories =
        {
        "Convective": ["_override_replace_", ("EW","W"), ("SV","W"), ("TO","W")],
        "Convective Watches": [ ("SV","A"), ("TO","A") ]
        }
```

The file HazardCategories.py is subject to is subject to incremental override. This means that when a list in an override is in the same namespace as a list in the base, the default behavior is to combine the override list with base list. So to remove items from an existing list in the base, one needs to either specifically remove those items (first example) or first stipulate that the base list should be ignored before presenting the override list (second example).

## Hazard Metadata

The Hazard Metadata is user-configurable per Hazard Type and appears in the Hazard Information Dialog. The configuration for the metadata consists of these types of files:

- HazardMetaData.py : One Python file which designates the metadata file name for each hazard type (or set of hazard types that share common metadata definitions). The code block below contains the default content of *HazardMetaData.py* so you can see the format
- Metadata Python files which contain the fields, choices, and display format for all the metadata to appear in the Hazard Information Dialog and subsequently in the products. For example, the MetaData_FF_W_Convective.py file is shown in Appendix 3.

```
# HazardMetaData.py
HazardMetaData =[
        {"hazardTypes": [("FF", "W", "Convective")],
          "classMetaData": "MetaData_FF_W_Convective"},
        {"hazardTypes": [("FF", "W", "NonConvective")],
         "classMetaData": "MetaData_FF_W_NonConvective"},
        {"hazardTypes": [("FF", "W", "BurnScar")],
         "classMetaData": "MetaData_FF_W_BurnScar"},
        {"hazardTypes": [("FA", "Y")], "classMetaData": "MetaData_FA_Y"},
        {"hazardTypes": [("FA", "W")], "classMetaData": "MetaData_FA_W"},
        {"hazardTypes": [("FF", "A")], "classMetaData": "MetaData_FF_A"},
        {"hazardTypes": [("FA", "A")], "classMetaData": "MetaData_FA_A"},
        {"hazardTypes": [("FL", "A")], "classMetaData": "MetaData_FL_A",},
        {"hazardTypes": [("FL", "W")], "classMetaData": "MetaData_FL_W"},
        {"hazardTypes": [("FL", "Y")], "classMetaData": "MetaData_FL_Y"},
        {"hazardTypes": [("HY", "O")], "classMetaData": "MetaData_HY_O"},
        {"hazardTypes": [("HY", "S")], "classMetaData": "MetaData_HY_S"},
  ]
```

Hazard Metadata is specified as megawidgets so that they can be displayed easily in the Hazard Information Dialog. For information on megawidgets see Chapter 2.

Overriding metadata files. Using the class-based approach, the override files need only contain the methods that need to be changed or added. As a simple example, suppose one wanted to have a more descriptive GUI label for one Call to Action for the FF.W.Convective hazard. The override file would need only contain:

```
# MetaData_FF_W_Convective.py:
class MetaData(CommonMetaData.MetaData):

    def ctaStayAway(self):
        return {"identifier": "stayAwayCTA",
            "displayString": "Stay away or be swept away",
            "productString":
            '''Stay away or be swept away. River banks and culverts can become
            unstable and unsafe.'''}
```

The green is what was added to the default instance of this method. Note that the default instance of this method is actually in *CommonMetaData.py*. By implementing this in MetaData_FF_W_Convective.py, this change only impacts that hazard subype. Were this an override to *CommonMetaData.py*, then every hazard type/subtype that made use of this CTA would see this change.

## Hazard Type Color Table

This functionality will eventually interoperate with the GFE Hazard Grid color table.

## Hazard Expiration Alerts

Focal points can configure when and how alerts based on hazard expiration manifest themselves. The base configuration is shown here. Please refer to that document in the discussion below. At a later time, there may be a GUI available to assist in this process. For now, other versions of this configuration can be created via the Localization Perspective.

Alerts are configured based on categories. In the base configuration, each hazard type belongs to its own category. However, site, etc. configurations can combined hazard types into other named categories. Hazard types are expressed by sub-field phenomenon (e.g. FA), significance (e.g. W) and, optionally, subtype (e.g. Convective).

For each category, a collection of named alert criteria are defined. The manifestations currently supported are in the console or as pop-up alerts. The units of the expirationTime can be hours, minutes or percent_completed. Percent_completed indicates how far along the hazard has progressed between the time it is issued and the expiration time of the earliest product generated by the hazard. So if a hazard is issued at 7:00am and the earliest product expiration time is 11:00am and the configuration is 25% then the alert will go off at 8:00am.

For Console (count-down-timer) alerts, the desired color is specified in terms of red/green/blue fractions between 0 and 1. So, for example, yellow would be specified as r=1.0, g=1.0, b=0.0. Various pages on the world-wide-web can help you define what those numbers should be to create the desired color. The color transparency (a=1.0) is ignored but should be left untouched. Optionally, whether or not the timer text is bold, blink and italic can also be specified via true/false. By default they are all false.

CHAPTER 3

# Megawidgets

Megawidgets are a set of graphical user interface (GUI) components found in use throughout Hazard Services. They provide an easy way for those who are modifying or building recommenders requiring user input to create dialog boxes in order to allow said recommenders to gather information from the user. Megawidgets are also used in other areas of Hazard Services that are configurable via localization: they populate the Details (metadata) section of the Hazard Information Dialog, and they may be used to create GUI components in the Product Staging Dialog for product generation as well.