

Focal Point Foundations

```
142 # ...
143 # ...
144 # ...
145 # ...
146 HOURS = 3600000
147 MINUTES = 60000
148 InclusionThresholdWarningDefault = False
149 OVERRIDE_LOCK = ['headline', 'combinationtags', 'jo
150 HazardTypes = {
151     'AF.V' : { 'headline': 'ADFWALL ADVISORY',
152               'override lock': OVERRIDE_LOCK,
153               'combinationtags': True,
154               'includeall': True,
155               'allowAreaChange': True,
156               'allowZoneChange': True,
157               'expirationTime': (-30, 30),
158               'endingExpirationDuration': 60 * MINUTES,
159               'endingExpirationThreshold': 15 * MINUTES,
```

Click "Next" to begin the presentation...

[No audio for this slide]

**Warning Decision Training
Division**

Presenter:

Eric Jacobsen

Course Contacts:

Eric Jacobsen

eric.p.jacobsen@noaa.gov



Hazard Services,
Focal Point Foundations Course

Tools & Recommenders

Welcome to the module covering “Tools & Recommenders”, part of the Hazard Services Foundational Training Course for Focal Points.

My name is Eric Jacobsen, with the Warning Decision Training Division. If you have questions about this course, or technical problems, please use the contact information listed on this slide.

Tools & Recommenders: Objectives

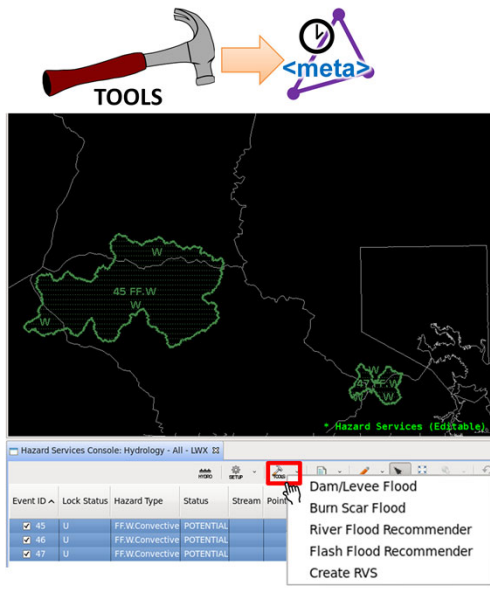
After completing this module, you will be able to identify:

- The **primary function** of tools and recommenders
- **Where** tools and recommenders are organized
- The **relationship** between **recommenders** and **tools**
- The **language used** for tools and recommenders
- The degree of **configuration possible**
- How recommenders primarily **access AWIPS data**
- **Limitations** in recommender **data access**
- **How visibility** of recommenders is managed

These are the objectives for this module. Please take a moment to review them, then, when you're done, click next to proceed with the module.

What are Tools & Recommenders?

- Suggest and modify **hazard events**
 - Type
 - Space and Time
 - Metadata
- **Code-based** path to hazard creation
 - Typically edited/completed by forecaster
 - Wide range of control



Tools and recommenders are one component of hazards services that's sure to expand in usefulness as the software takes hold.

As was summarized in the workflow segment, tools and recommenders assist forecasters by initiating and managing hazard events. This may entail setting the hazard types, managing their spatial and temporal extents, and even pre-populating some of the hazard attributes which we know as Metadata.


Ultimately, tools represent an alternate, code-based path to hazard creation which parallels or surpasses some of manual capabilities users have access to. Typically, the generated events may be interacted with by forecasters through the console and Hazard Information Dialogue before product generation.

However – as we'll see later – tools are not limited to specific user actions and have a wide range of control behind the scenes, to where they may even, in exceptional cases, send events directly to product generation with minimal user interaction.

Let's take a closer look at the design and power of tools.

Currently Available Tools & Recs

Name	Hazards	Data	General Configuration	Likelihood
Flash Flood Recommender	FF.W, FF.A	FFMP	None (inherits config from FFMP)	Low
River Flood Recommender	FL.W, FL.A, FLY, HYS	Hydro	Code: Potentially remove hazard types Other: Reformat some river flood products*	Possible
DamLeveeFlood	FF.W, FF.A <i>nonConvective</i>	Maps	Other: Provide shapefile and scenario details	Required
BurnScarFlood	FF.W, FF.A <i>nonConvective</i>	Maps	Other: Provide shapefile and burn details	Required
RVS Tool	n/a	Hydro	Other: Potentially reformat river stage product*	Possible

 As of Hydro IOC, Aug 2019

- Baseline hazards, data, and setup
- Configuration preview:
 - Often relate *to supporting files, follow-up processes*
 - Tool logic does not need substantial change

Before we begin, this chart presents a summary of the tools and recommenders which are currently available, valid for the Hydro Initial Operating Capability.

Some core design features of each tool and recommender are summarized here for the benefit of focal points. For example, the second column contains the hazard types that may be generated by each. Likewise, the primary data types accessed by each tool are shown, which relates to the database tables that python code is actually querying via a framework introduced later in this module.

Perhaps most importantly, the final columns identify the more common configurations related to each tool that focal points may need to make (and their likelihood). But we'll revisit these columns of the table a little later, after taking a look at how tools and recommenders are set up.

For now, rest assured that, although small tweaks to code may be needed for minor changes to tool behavior, many configuration tasks will relate not to the tool's logic, which is functional as delivered, but to auxiliary files or follow-up processes. Substantial edits to code for tools and recommenders are very unlikely to be necessary.

Tools vs. Recommenders

Recommenders

- Create or update hazard events
- Source data -> “first guess”
- Pass to forecaster for consideration

Examples:

- River Flood Recommender
 - Make initial hazard recommendations based on river data

Tools

- Operate on hazard events or related data
- More clear-cut outcome

Examples:

- “Zorro” tool (*not IOC*)
 - Complex polygon masking
- “Create RVS” tool
 - Routine product generation

- Subtle differences in application/use
- Configurations share common framework and similar code

Before we get too far, is there a difference between tools and recommenders?

The differences, to be sure, are subtle, and mainly relate to their application, rather than their back-end construction. In fact, from a configuration standpoint the two are very similar, as we’ll see shortly that all tools and recommenders branch from the same templates and use very similar code.

Nevertheless, the following loose classification can be useful in explaining their differing applications.

Recommenders are focused on providing a “first guess” for a potential hazard (or multiple), which they do through comparing source data to user-defined parameters. Examples of this include the river flood recommender, which analyses the hydro database to identify possible river flood hazards, but which a forecaster subsequently validates prior to manually issuing any product.

In comparison, tools also operate on hazard events, but with a more clear-cut, generic outcome. For example, forecasters will eventually see advanced tools for masking complex polygons with specific conditions. Or there is the RVS tool, which – though it does dynamically query hydrologic data – is not interactive and directly produces a routine RVS product.

Once again, from a configuration standpoint this distinction is not extremely rigid, and so

we'll often use the terms "tools" and "recommenders" interchangeably in the rest of this module.

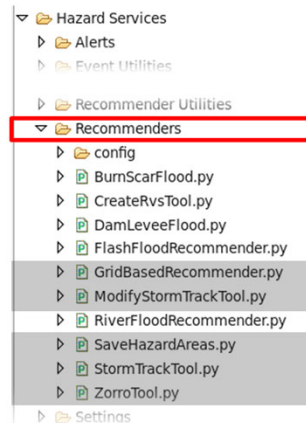
How they work

INSIDE TOOLS & RECOMMENDERS

Let's take a closer look now at the construction and inner workings of tools and recommenders.

Recommenders in the Localization Perspective

- Localization Perspective > Hazard Services > Recommenders
 - **config**: Core framework files
 - One file for each tool/recommender
 - Additional future tools, currently disabled



Focal points should know that all tool and recommender files, which we're about to look at, are available for viewing – and editing - in the localization perspective, under the “Recommenders” directory within “Hazard Services.”

What you see within this folder will include; a “config” subdirectory, mainly for core framework files, which should not be edited; and a file for each tool or recommender.

Some visible tools may not yet have been deployed operationally, and their use will be disabled by settings. For now, this module only addresses those which belong to hydro IOC

Tool/Recommender Logic



- **Fully Editable** (as most things in HazSvc)
- **Python class** based on “recommender” template
- Built from **methods** which act on inputs to yield hazard event information
- **Interfaces** available for connecting to AWIPS datasets
 - E.g. Data Access Framework

```
class Recommender(RecommenderTemplate.Recommender):
    def __init__(self):
        self._riverProFloodRecommender = None
        self.bridge = Bridge()
        self._riverForecastUtils = RiverForecastUtils()
        self._riverForecastManager = RiverForecastManager()
        self.hazardEventLockUtils = None
        self.siteId = None
        self.tpc = TextProductCommon()

    def execute(self, eventSet, dialogInputMap, spatialInputMap):
        """
        Runs the River Flood Recommender tool

        @param eventSet: A set of events which include session attributes
        @param dialogInputMap: A map of information retrieved from a user's interaction with a dialog
        @param spatialInputMap: A map of information retrieved from the user's interactive spatial display.

        @return: A list of potential events.
        """
        sessionAttributes = eventSet.getAttributes()
        sessionMap = JUtil.pyDictToJavaMap(sessionAttributes)
        if self.hazardEventLockUtils is None:
            practice = GeneralUtilities.isPractice(sessionMap)
            self.hazardEventLockUtils = HazardEventLockUtils()
        if self.siteId is None:
```

Tools and Recommenders are written in Python. Existing ones can be edited and new ones created.

Behind the scenes, they are simply class files, which –as explained in the Python overview – suggests that they are like templates, in this case for repeatable actions, as tools and recommenders certainly are.

All tools and recommenders also inherit functionality from a parent class in the RecommenderTemplate file, which simply outlines the structure for writing new recommender code.

Recommenders heavily use internally defined python “methods,” such as the “execute” method shown here on screen, to accomplish their tasks. Ultimately these functions act on inputs (which can vary depending on the recommender or scenario) to yield hazard event information.

Another pivotal aspect of recommenders is their ability to interface with AWIPS datasets, in particular through the data access framework which will be covered later.

Recommender Inputs

Functional Inputs

Session: Current session information

Ex: what is CAVE clock time?

Input Dialog: what options should guide how the recommender is run?

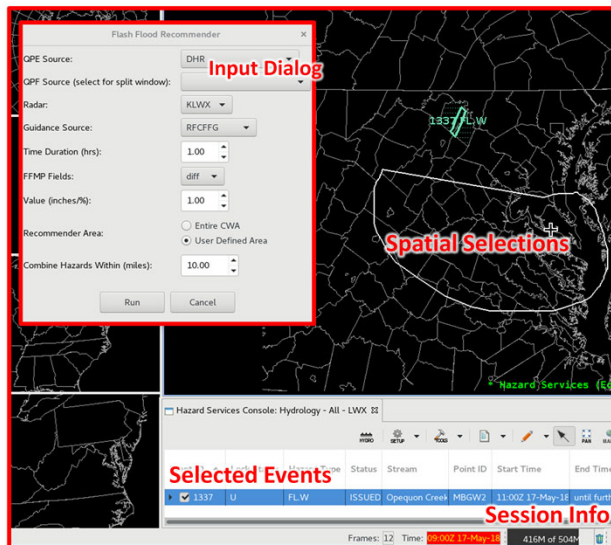
Ex: what QPE inputs for flash flooding?

Spatial Selections: are point, polygon, or other spatial interactions used?

Ex: point for storm track

Hazard Events: recommender can be triggered to run with one or more hazards selected

Ex: Masking polygon with edit areas



Although an in-depth review of Recommender code is beyond the scope of this training, focal points who are interested in configuring recommenders should be familiar with the basic inputs, and how some of them correspond to key methods within the file.

From a functional perspective, recommender inputs can include: information about the session, such as the CAVE clock time; a configurable input dialogue, consisting of execution options and criteria; potential spatial selections such as points or polygons drawn; and finally, hazard events which may have been selected prior to running the recommender.

Recommender Code Components

```
def defineDialog(self, eventSet, **kwargs):
    """
    @return: A dialog definition to solicit user input be
    """
    methodInput = None
    jMethodInput = kwargs.get("methodInput", None)
    selectedPointID = None
    if jMethodInput is not None:
        methodInput = JUtil.javaMapToPyDict(jMethodInput)
        selectedPointID = methodInput.get('selectedPointID')
        self.siteId = methodInput.get('siteID')
    dialogDict = {"title": "Flood Recommender"}
```

```
def execute(self, eventSet, dialogInputMap, spatialInput
    """
    Runs the River Flood Recommender tool
    @param eventSet: A set of events which include sessi
        attributes
    @param dialogInputMap: A map of information retrieve
        a user's interaction with a c
    @param spatialInputMap: A map of information retri
        from the user's interacti
        spatial display.
    @return: A list of potential events.
    """
    sessionAttributes = eventSet.getAttributes()
    sessionMap = JUtil.pyDictToJavaMap(sessionAttributes)
    if self.hazardEventLockUtils is None:
        practice = GeneralUtilities.isPractice(sessionAt
```

return filteredEventSet

Core Code Components

__init__

- Basic parameter initializations

*defineScriptMetadata

- Reference info about code & authorship

defineDialog

- Megawidget collection for building options GUI prior to execution

defineSpatialInfo

- Define what spatial display input, if any, is used by recommender

*execute

- CORE function of recommender
- consolidates ALL inputs (dialog, spatial info, event sets), returns event set

** required*

Certain of these inputs correspond to well-defined methods within a recommender's code which can be modified by the focal point.

Notably, the "defineDialog" method is the method which constructs the recommender's input dialogue, such as the one shown here for setting how the flash flood recommender will run before it executes. These dialogues, like many others in Hazard Services, are built with megawidgets (which were introduced in the metadata & megawidgets section). Edits here can change the behavior, choices, defaults, included megawidgets, and more for recommender dialogues.

Another method, "defineSpatialInfo" can be used to specify what, if any, spatial selections are supported by the recommender. For example, whether to consider only a user-selected area (as shown), a specific point, or some other criteria.

The method at the heart of the recommender, however, is "execute." Every recommender must have this method, which consolidates all inputs and returns an event set. Focal points editing existing recommenders or writing their own will concentrate much of their customizations on the behavior within this "execute" method and any auxiliary methods which help it.

Ultimately, the "execute" method always ends in the output of events, if any were found, which embodies that the fundamental focus of all tools and recommenders is on manipulating events.

The Hazard Event “API”

```


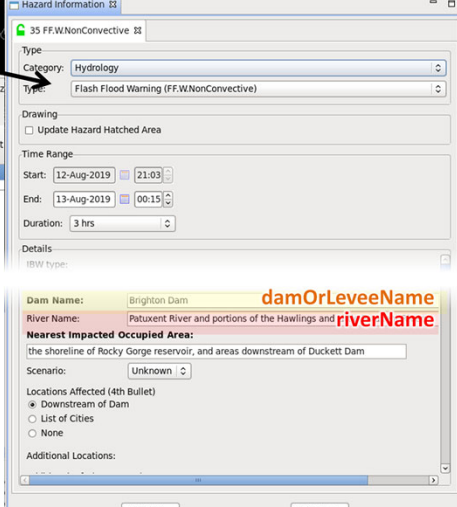
DamLeveeFlood.py
if hazardEvent.getEventID() == None:
    hazardEvent.setEventID("")
    hazardEvent.setHazardStatus("PENDING")

hazardEvent.setSiteID(str(sessionDict["siteID"]))
hazardEvent.setPhenomenon("FF")
hazardEvent.setSignificance(significance)
hazardEvent.setSubType(subType)

hazardEvent.setGeometry(GeometryFactory.createCollection([hazardGeometry])
                        if hazardGeometry else None)
hazardEvent.setHazardAttributes({
    "cause": "Dam Failure",
    "damOrLeveeName": damOrLeveeName,
    HazardConstants.RECOMMENDED_EVENT: True,
    "riverName": riverName,
    "preDeterminedArea": True
})

return hazardEvent

```

- Tools & Recommenders *directly access* hazard events
 - Virtually full control for setting any attribute, value
 - Event methods similar to “API”
- Assignments are “hard-wired”, self-contained

Although the exact ‘wiring’ of each recommender and tool may be very different, what’s inherent to them all is an ability to directly access hazard creation and attribute assignment.

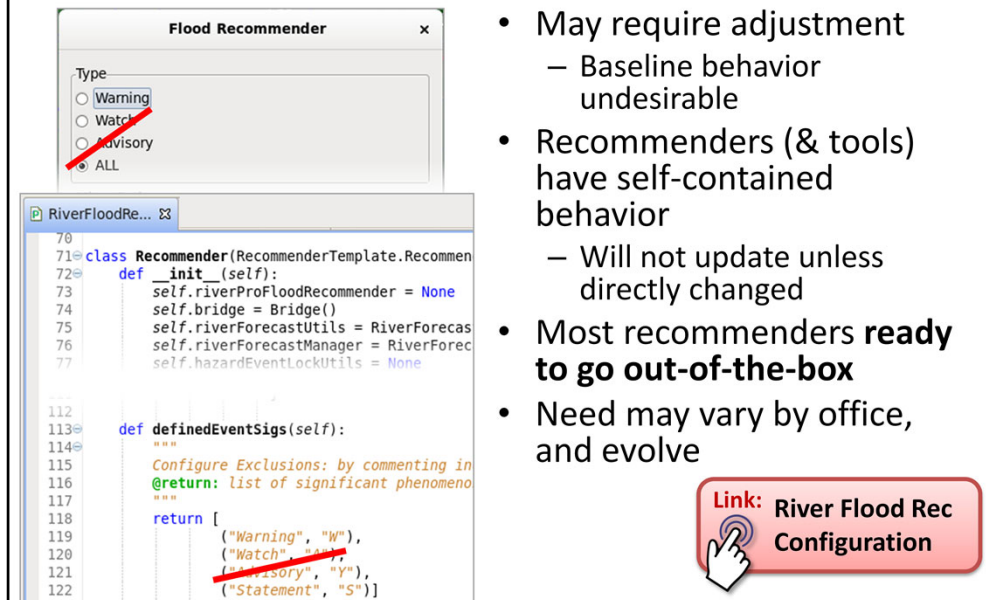
Just to help illustrate this point, consider the relatively lightweight - but still capable - dam break recommender. This tool takes user selection of a pre-configured dam, as well as an “urgency” level, and initiates a hazard of matching significance, using a polygon from the maps database for that dam, and launches the HID with certain pre-populated megawidget fields.

Through this process, many of the manual steps that the user might have taken in the spatial display, and in the HID, to craft a hazard were executed behind the scenes before the HID was even shown. This was enabled by a set of powerful methods granting control over “hazard events”... something like what programmers might call an “Application Programming Interface”, or “API”.

Peering more closely at the code for the dam break recommender, we get a glimpse of some of these methods, such as: the assignment of “FF” as the phenomenon; one of two significance types (watch or warning) –read from the user input on the dialog; the hazard geometry; and also a hazard cause and other helpful “attributes” which, where they happen to correspond to the “field name” of a megawidget that later appears on the HID, will cause those fields to be pre-populated.

The syntax of these methods is not as important for beginning focal points to memorize as it is to appreciate that these routines are, in a sense, “hard-wired” to generate certain hazard types and metadata. And, moreover, they execute their task in a self-contained way with no direct awareness of the constraints of the HID.

Manage Baseline Recommenders



- May require adjustment
 - Baseline behavior undesirable
- Recommenders (& tools) have self-contained behavior
 - Will not update unless directly changed
- Most recommenders **ready to go out-of-the-box**
- Need may vary by office, and evolve

[Link: River Flood Rec Configuration](#)

A time may come where focal point need to edit some parts of baseline recommenders, for example if the hazard choices or options they present are undesirable. Being largely self-contained scripts, the hazards and metadata that tools and recommenders suggest must be directly managed within their code to effect any change.

Many recommenders are fine out-of-the-box for some offices. For example, the baseline river flood recommender (shown), as well as the flash flood recommender and RVS tool, are provided ready to use as long as offices don't have different product issuance policies.

To this last point, the river flood recommender could be one tool which offices need to adjust, say to remove the ability to issue anything except for warnings. This would involve editing choices in the inputDialogue method and ensuring that other event assignment processes throughout the code have been completely decommissioned which consider non-warning events.

As the number of tools grows and offices move to more advanced use of Hazard Services, a greater familiarity with recommender code will be very beneficial to focal points seeking to make such configurations.

Configuring Tools for Pre-Defined Impact Areas

The screenshot displays a software interface for configuring tools for pre-defined impact areas. It includes a dialog box for 'Dam/Levee Flood', a table of dam data, a file browser showing utility files, and a metadata configuration window. A red button labeled 'Link: Dam/Burnscar Configuration' is highlighted with a hand cursor.

gid	name	the_geom	cwa	type
[PK]	character varying(80)	geometry(MultiPc	character	character varying(
1	Lake Barcroft Dam	0106000020E61000(LWX		daminundation
2	Upper Occoquan Dam	0106000020E61000(LWX		daminundation
3	Fairview Lake Dam	0106000020E61000(LWX		daminundation
4	Huntsman Lake Dam	0106000020E61000(LWX		daminundation
5	Lake Accotink Dam	0106000020E61000(LWX		daminundation
6	Lake Barton Dam	0106000020E61000(LWX		daminundation
7	Lake Holiday Dam	0106000020E61000(LWX		daminundation
8	Savage River Dam	0106000020E61000(LWX		daminundation
9	Jennings Randolph Dam	0106000020E61000(LWX		daminundation
10	Duckett Dam	0106000020E61000(LWX		daminundation
11	Laurel Lumber Dam	0106000020E61000(LWX		daminundation
12	Brighton Dam	0106000020E61000(LWX		daminundation

```

DamMetaData = {
  "Lake Barcroft Dam": {
    "riverName": "portions of Holmes R",
    "damName": "Lake Barcroft Dam",
    "cityInfo": "between Lake Barcroft",
    "ruleofthumb": "Areas between Lake",
    "dropDownLabel": "Lake Barcroft Dam",
    "featureID": "LakeBarcroftDam"
  },
  "Upper Occoquan Dam": {
    "riverName": "Occoquan River",
    "damName": "Upper Occoquan Dam",
    "cityInfo": "Occoquan",
    "ruleofthumb": "Interests in low l",
    "dropDownLabel": "Occoquan Dam (Fa",
    "featureID": "OccoquanDam"
  },
  "Fairview Lake Dam": {
    "riverName": "portions of Holmes R
  
```

- Pre-defined area tools (IOC):
 - Dam/Levee Flood
 - BurnScar Flood
- Required elements:
 - Maps: hazardServicesArea
 - MetaData files in Utilities
- Scripts provided to:
 - Migrate from WarnGen
 - Add/Update

Link: Dam/Burnscar Configuration

One type of tool will require much more Focal Point involvement both initially and over time. These are the tools used for “pre-defined impact areas”, or locations with a known risk which have been encoded in advance.

In particular, two IOC tools –one for dam or levee breaks, and another for burnscar flash flood products – depend on a pre-populated map table and utility files to allow any selection in their initial dialogues. In fact, the intensive configuration we’re referring to will be in populating these supporting pieces, and not to the code itself, which is ready to work as soon as back-end data has been made available.

The specific maps table is called “hazardserviecsarea” and is previewed here. This single table is intended to store all pre-defined areas, whether dam, burnscar, or even some custom type, and a “type” column is provided to differentiate each group.

In addition, each tool will have its own file for additional “metadata,” saved under the “utilities” folder of Hazard Services, which allows for pre-determined location information, scenario text, and more to be saved and called on by the tool for attributing to the hazard event and populating the HID.

The easiest way to create the records and files referred to here is if offices already have this data in WarnGen. Conversion scripts are provided with Hazard Services, whose use is documented in the linked jobsheet, for migrating data from old velocity and xml files to the new structure. In addition, the same scripts and jobsheet may be used for new records to

import shapefiles or to add additional information.

After their initial setup, focal points should plan to maintain this table and files with up-to-date information to be best prepared to call on them during events.

Configuration Overview for Current Tools

Name	Hazards	Data	General Configuration	Likelihood
Flash Flood Recommender	FF.W, FF.A	FFMP	None (inherits config from FFMP)	Low
River Flood Recommender	FL.W, FL.A, FLY, HYS	Hydro	Code: Potentially remove hazard types Other: Reformat some river flood products*	Possible
DamLeveeFlood	FF.W, FF.A <i>nonConvective</i>	Maps	Other: Provide shapefile and scenario details	Required
BurnScarFlood	FF.W, FF.A <i>nonConvective</i>	Maps	Other: Provide shapefile and burn details	Required
RVS Tool	n/a	Hydro	Other: Potentially reformat river stage product*	Possible

- Most setup does not change **tool** logic
 - **“Other”**: Supporting files or follow-up processes
- Varying need for initial configuration
- *Associated product changes are separate from recommender code
- Anything (GUI to logic) is editable

 As of Hydro IOC, Aug 2019

Armed with a better understanding of their operation, let’s return to the table showing configuration needs for currently available tools and recommenders.

Jumping to a key takeaway, many of the configuration needs depicted on this slide do NOT entail editing the logic within tool or recommender code. Configurations which are related to recommenders, but which involve some external file or process, are labeled “Other” in this table.

Little to no change may be needed for some tools, such as for the “Flash Flood Recommender,” which in fact leans mostly on the FFMP setup to populate its fields.

Others may function out-of-the-box, but have associated behaviors which are undesirable for some offices. The “RVS tool” and “River Flood Recommender” fall into this group, due to the hazards they issue or the products they lead to.

Relatively superficial edits to the River Flood Recommender’s logic are all that’s needed to change the generated hazards. However, it’s completely external to recommender code to tailor the product formats that might be triggered downstream of them, owing to Hazard Services fundamentally separate treatment of hazards (to which recommenders contribute) and products.

Finally we’ve seen that some tools, such as the the Dam and Burnscar tools in Hydro IOC, will provide no useful output until local shapefiles and details on which they operate are

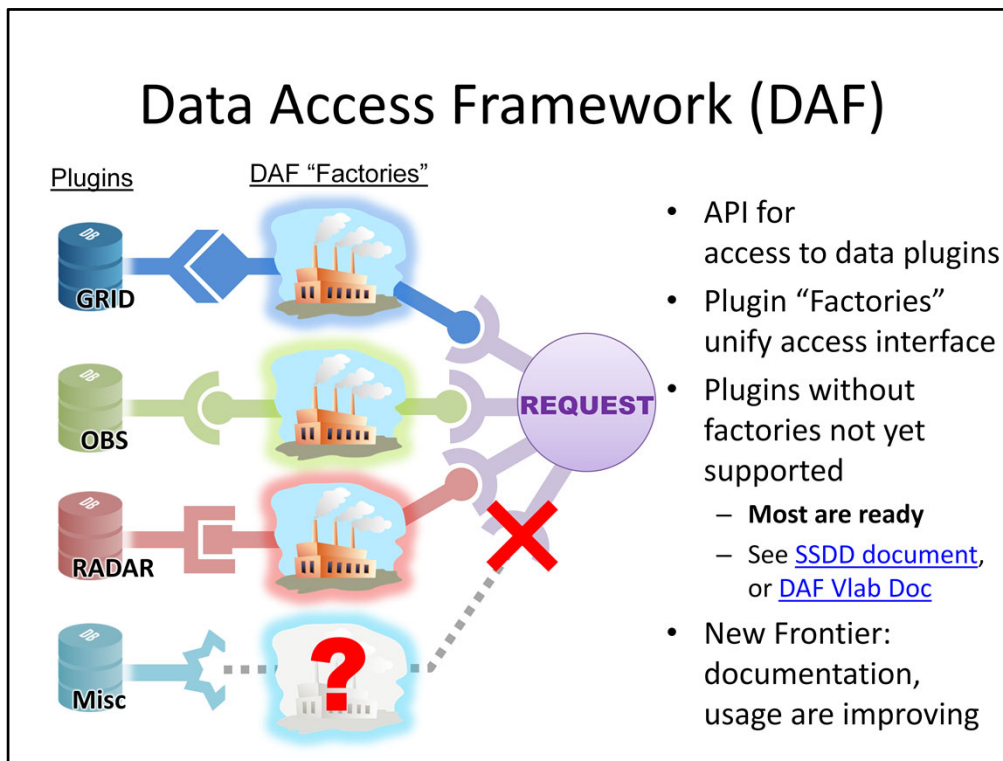
contributed by the focal point. Though a substantial effort, it IS important to identify once again that this configuration deals with laying the groundwork in “*auxiliary*” files, and NOT in the actual python code itself, which is fully functional and very unlikely to need edits.

Although most common configurations may not require substantial edits to the recommender code, recall that, like so much in Hazard Services, all aspects of tools and recommenders are fully editable if desired.

Accessing AWIPS Data

THE DATA ACCESS FRAMEWORK

In some of these last slides, we'll introduce another important python framework at the heart of many tools.



The decision-making that recommenders are tasked with is critically dependent on the AWIPS Data Access Framework, often abbreviated as “DAF.”

Every AWIPS data type, equivalently termed plugins, has a unique way of being stored and accessed. The DAF is an attempt to unify and simplify the commands needed to access all plugins.

Although a bit technical-sounding, so-called “factories” can be programmed for each plugin to map the normally complex handling of each data type to the common commands used by the DAF. Over numerous plugins, this creates a consistent and comparatively easy framework for what would otherwise be an overwhelming mix of behaviors.

The DAF, however, depends on these factories being written for each plugin before they can be accessed, and although a few plugins may not yet have factories, the vast majority of useful types do and can be accessed via the DAF. The Software System Design Document or a community-created DAF VLab page can both be referenced to see the data types available for use, and to get a start on writing DAF commands.

DAF in some ways remains a new frontier, with an ongoing effort by its growing community to more fully document and test it. It may still be possible to encounter some “rough edges” in the meantime, but for most applications it is a powerful resource which is ready to use.

DAF Code

- DAF used extensively in Hazard Services Recommenders and Tools
- Python API usage
 - “Request”-centric
 - Relies on DataAccessLayer library
 - Common request syntax and return formats regardless of type
- Small toolkit for broad data access



Excerpt from Flash Flood Recommender

```
def getQPEValues(self):
    """
    Strategy method
    from preprocess
    """
    request = DataAccessLayer.newDataRequest()
    request.setDatatype(FRRP_KEY)
    request.setParameters(self._qpeSourceName)
    request.addIdentifier(WFO_KEY, self._currentSite)
    request.addIdentifier(SITE_KEY, self._siteKey)
    request.addIdentifier(DATA_KEY, self._dataKey)
    request.addIdentifier(HUC_KEY, ALL_HUC)
    availableTimes = DataAccessLayer.getAvailableTimes(request)
    usedTimes = []

    startTime = self._currentTime - (self._accumulationHours * GeneralConst)
    endTime = self._currentTime

    if self._splitWindow:
        startTime += self._splitWindowMillis

    for time in availableTimes:
        refTimeMillis = time.getRefTime().getTime()
        if refTimeMillis >= startTime and refTimeMillis < endTime:
            usedTimes.append(time)

    self._qpeGapInfo = self._getGapInfo(self._dataKey, self._qpeSourceName,
    basins = [])
    if usedTimes:
        geometries = DataAccessLayer.getGeometryData(request, usedTimes)
        for geometry in geometries:
            self._storeQpe(geometry.getLocationName(), geometry.getNumb)
        return True
    return False

def getQPFValues(self):
```

In Hazard Services, the python DAF interface is used extensively in recommenders, and also for other purposes where data access is needed. An example use of the DAF is shown here from the Flash Flood Recommender code, specifically within the method for getting QPE values to help determine the precipitation received over the relevant area.

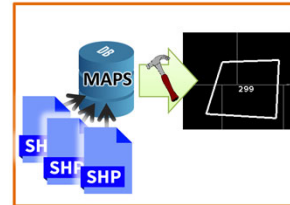
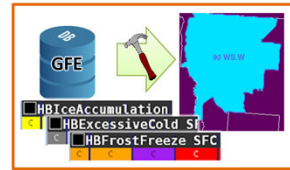
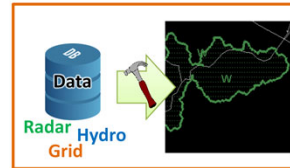
Fundamentally, using the DAF consists of building a data “request, initiated using the “DataAccessLayer” python library, which is at the heart of DAF use in HazSvcs python files. All DAF requests are similarly constructed, regardless of the data type, with only a few common methods, including “setParameter” and “addIdentifier” used here. These methods take arguments specific to the data type which, in combination, help to narrow the request.

Once sufficiently specific, the request can be used by various methods to retrieve data, for example: the list of available times for the requested parameter; or geometric data, such as shapes and points and corresponding values, which in this particular case contains the qpe values mapped to their geographic coordinates, as visualized here. Other retrieval options are available, such as to get gridded data, but ultimately the beauty of DAF lies in the limited number of commands necessary for accessing data of any supported type.

Focal points are encouraged to check out the available references to better understand the construction and return of DAF requests, but this example was shown to give a general sense of how such requests appear and are used within HazSvcs python.

DAF Uses

- Examples of DAF usage with AWIPS:
 - Retrieve **grid** or other **observation data** from databases and compare to hazard thresholds
 - Query **GFE** grids to identify forecast hazards
 - Load pre-defined **shapes** from **maps database** for known impact area (e.g. dam breaks, burn scars)



While potential uses for the DAF are nearly limitless, a few common applications are summarized here.

The DAF can be used within recommenders or other scripts to retrieve grid or other observation data from AWIPS, which can then be analyzed through Python code to detect potential hazards

One of the many supported plugin types is GFE grids, thus enabling the DAF to retrieve and analyze forecast data for potential risks

Other databases accessible by DAF include maps. Loading pre-defined polygons in the maps database with shape-files, such as for dams and burn scars, then retrieving these with the DAF, supports rapid, accurate hazard creation for pre-defined impact areas.

As Hazard Services matures, expect more discussion and guidance on the uses for DAF, especially with new hazard types

“Registering” Tools with Settings

- Accessibility within Settings
 - ALL recommenders visible by default
 - Exception: restricted localization level
 - **TIP:** Keep HazSvcs development code private, not site-wide
- “Register” tools
 - CommonSettings.py
 - Prevents AlertViz errors

Settings GUI accessed via console
Edit Setting: Hydrology_All

Name: Hydrology_All
Display Name: Hydrology - All
Category: Hydrology

Hazards Filter Console Console Coloring HID/Spatial **Recommenders** Ma

Available Recommenders:
StormTrackTool
MyExperimentalRecommender

Visible Recommenders:
DamLeveeFlood
BurnScarFlood
RiverFloodRecommender

Author: Focal Point
Version: 1.0
Description: Just testing - I don't want anyone to see this yet! **!**

Save Save As ... Reset Dismiss

```
CommonSettings.py
170 "toolbarDefinitions": {
171   "DamLeveeFlood": {
172     "displayName": "Dam/Levee Flood",
173     "toolType": "USER_RECOMMENDER",
174   },
175   "BurnScarFlood": {
176     "displayName": "Burn Scar Flood",
177     "toolType": "USER_RECOMMENDER",
178   },
179   "RiverFloodRecommender": {
180     "displayName": "River Flood Recommender",
181     "toolType": "USER_RECOMMENDER",
182   }
183 }
```

One final configuration point for recommenders overlaps with managing the Hazard Services settings. This deals with how tools and recommenders become accessible to a user, in particular within the settings management GUI.

All files in the recommenders directory, in fact, will by default be automatically visible to the settings interface in the “Recommenders” tab, except for those in more restricted localization levels. This emphasizes the importance of testing new recommenders, and Hazard Services configurations in general, on a user- or more isolated localization level, to prevent confusion in operations, even when running CAVE in practice mode.

With tools and recommenders, it is also important, however, to “register” the tool in the “common settings” file under the “toolbarDefinitions” variable, or else an alertviz message will appear indicating ambiguity about the tool. Jobsheets for creating new recommenders will include the appropriate, straightforward steps for modifying this variable.

Tools & Recommenders Takeaways

- Generate/manage **hazard events** & metadata
 - Wide range of flexibility
- Recommenders and Tools similar (subtle differences)
 - **Common templates** and code
- **Fully editable, Python** code modules
 - Accessible in “**Recommenders**” **directory** under “Hazard Services” in Localization Perspective
 - Need to “**register**” **new tools** in common settings

To summarize tools and recommenders in Hazard Services, we emphasize that their fundamental purpose is to generate and manage hazard events and their metadata. A very capable set of event access methods enable a wide range of flexibility in this area

We’ve clarified that Recommenders and Tools serve subtly different roles in building events, but are structurally quite similar behind the scenes. This is especially reflected in the fact that they share identical templates for their construction, and leverage very similar code.

Speaking of code... like so much in hazard services, tools and recommenders are written in python and are fully editable, allowing existing ones to be modified and new ones to be created. All recommenders and tools (and their supporting files) can be seen and edited in the “Recommenders” directory within the Localization Perspective under Hazard Services’

We’ve also noted that, once offices do begin to create their own tools, it’s important to “register” them with the common settings file.

Tools & Recommenders Takeaways, 2

- Almost **complete configurability**
 - Most recommenders are **ready-to-go**
 - IOC: *Flash Flood, RVS, River Flood recommenders*
 - Pre-defined “**Impact Area**” tools require **significant setup** and maintenance
 - IOC: *Dam Break, Burn Scar tools ([Jobsheet](#))*
- Powerful **Data Access Framework**
 - Maps + **most AWIPS data types can be accessed** within tools and recommenders
 - Documentation and experience still improving

Thanks to their construction in editable Python, tools and recommenders can be almost completely re-configured. With that said, most will be ready-to-go and can be used effectively without any edits.

In hydro initial operating capability, this includes the Flash Flood recommender, RVS tool, and River Flood Recommender, although it's likely that offices may need to configure especially the River Flood recommender to limit the hazards it may issue out-of-the-box.

Tools which rely on a database of 'pre-defined impact areas', however, which are unique to each CWA, will require significant initial setup and maintenance to be effective. In IOC, this includes the dam break and burn scar tools, whose setup is documented in the provided jobsheet.

Finally, one of the more powerful, configurable elements of tools and recommenders is the data access framework, which allows access to maps, grids, and most other useful AWIPS data types for a wide variety of analysis and decision making. As the user base and documentation continues to grow, this should help to refine the overall experience of leveraging this very powerful framework.

Take the Quiz



You're almost finished!

Click "**Next**" to take the quiz.

[No audio for this slide]



Tools and Recommenders Assessment

Quiz - 9 questions

Last Modified: Aug 27, 2019 at 08:15 AM

PROPERTIES

On passing, 'Finish' button: [Goes to Next Slide](#)

On failing, 'Finish' button: [Goes to Next Slide](#)

Allow user to leave quiz: [After user has completed quiz](#)

User may view slides after quiz: [At any time](#)

Show in menu as: [Single item](#)

