**Background**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions#Background)

* awips2 aims to be a blackbox re-write of awips1. It is written mostly in java, but python is extensively used too, esp in gfe. awips2 is built upon large, thriving open source projects
  + eclipse: <http://www.eclipse.org>
  + camel: <http://camel.apache.org/>
  + spring: <http://www.springsource.org/>
  + hibernate: <http://www.hibernate.org/>
  + qpid: <http://qpid.apache.org>
  + others - postgres, HDF5, log4j, thrift
* Common AWIPS Visualization Environment (CAVE)
  + will run on the workstations, like d2d, gfe, and hydro do today
  + CAVE is an RCP application. The RCP is eclipse's Rich Client Platform. The RCP is a framework of a core set of ~30 plugins. See <http://wiki.eclipse.org/RCP>
  + plugins are the smallest unit of RCP functionality. With plugins code can be loosely coupled, maintainable and extensible
  + these core plugins provide "extension points", which can be used by applications to extend the funtionality of the framework
  + one such extension point is org.eclise.ui.perspectives. In awips2, d2d, gfe, and hydro are cave perspectives
  + CAVE variations
    - the map pans and zooms
    - no menu tearoffs, but views are detachable
    - a localization perspective exists. It's from the localization perspective that smartTools and Utilities are edited, and not by right-clicking in the EA window like in awips1
    - gfe perspective
      * no GM realignment button. Realign the GM by dragging
    - ... what else?
* the server is EDEX: Environmental(?) Data EXchange. EDEX consists of
  + an Enterprise Service Bus (ESB)
    - JVMs. There are currently 4 per server, clustered across 2 servers. Spring and camel form the backbone of the ESB. It's in this environment that awips2 server code runs.
      * [spring](http://www.springsource.org) manages software objects. It creates them (instatiates them), and injects them with initial values using directives retrieved from xml. Makes for a loosely coupled, flexible system. Change the runtime behavior of the system by modifying the xml, not the java source code. See <http://static.springsource.org/spring/docs/2.5.x/reference/beans.html>
      * [camel](http://camel.apache.org) routes messages using the Enterprise Integration Patterns described here <http://www.enterpriseintegrationpatterns.com/eaipatterns.html>
  + a JMS broker - messaging middleware.
    - awips2's JMS broker is [QPID](http://qpid.apache.org)
    - qpid is an important part of data ingest. The sbn ldm client, radar, and ldad ingest procs all send messages to a broker queue, which is read by the ESB.
    - once the message is read, the ESB looks in /awips2/edex/data/utility/edex\_static/base/distribution to determine which data plugin to use to ingest and decode the raw product (yes, edex has plugins too, although they are not RCP plugins). Each data type has a plugin (there are about 50), and each plugin has a data access object (a "dao") or a set of DAOs that is used to store and retrieve data from the database. The DAOs interact with postgres's metadata db and the hdf5 filesystem.
  + postgres - metadata database used in conjunction with the hdf5 filesystem to form the awips2 database. (With hydro being the notable exception. Hydro databases will be left as they are in awips1.)
  + HDF5 filesystem - data stored here. Not meant to be accessed directly. <http://www.hdfgroup.org/HDF5/>
  + *[say something about* [*pypies*](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/TimNotesTwo#WhatispyPIES)*? Not a part of the original design ...]*
* awips2 has a development environment called the Awips Development Environment (ADE). The ADE consists of eclipse's IDE (also an RCP app) and the software for the entire system, minus the legacy rehosted code. Clients like CAVE can be run from the ADE, so it's possible to make changes to CAVE code and run your changes. EDEX too. Make changes to EDEX, then compile and deploy with the ADE and run your modified EDEX
* [tips](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/HowToNewsFromRaytheon) from Raytheon

**Subversion**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions#Subversion)

By far, the most important single step you can take to learn subversion is to set aside some quiet time and read <https://collaborate.werh.noaa.gov/wiki/index.php/SubversionHelp>. It is written by Paul Jendrowski (RNK), and is an exceptional intro. Nothing that follows in this section is a substitute for reading Paul's page. See also [HowToSubversion](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/HowToSubversion) which should be the intro to Subversion starting point for local applications developers.

Two sites you should know about:

1. <https://collaborate.nws.noaa.gov/trac/ncladt> - wiki for local app developers. It has an svn repository where local apps will be stored.
2. <https://collaborate.nws.noaa.gov/trac/testtrac> - wiki for honing your [wiki formatting](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/WikiFormatting) skills. Practice here. This site also has an svn repository, with which we'll be practicing.

In the svn repository, ported local apps go in ladroot. See the [repoLayout](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/RepoLayout#RespositoryLayout) page. Browse to corresponding place with Browse Source

**a simple example**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions#asimpleexample)

The example below is really just for demo purposes. Its text will be used for cutting and pasting. As mentioned above, for a real intro, read <https://collaborate.werh.noaa.gov/wiki/index.php/SubversionHelp>

Scenario: open three windows. One we'll use to ping the svn server, and the other two will represent two different users: awips and jho. Then,

* user jho checks in an app. 'fakeapp' represents an awips1 app
* user jho tags it as the awips1 version
* user awips checks out fakeapp
* user awips modifies fakepp - ports it to awips2
* user awips checks in the modified fakeapp
* user awips tags the modified fakeapp as the initial awips2 version

Our local app (see the [local app guide](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/LocalApplicationsGuide) for local app guidelines in awips2)

mkdir /localapps/runtime/fakeapp

cd /localapps/runtime/fakeapp

vi fakeapp.txt ...

Check into subversion (see [HowToSubversionNewApp](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/HowToSubversionNewApp) for standardized instructions). First, let's query the server side

export REPO=https://collaborate.nws.noaa.gov/svn/testsvn/ladroot/apps

svn ls $REPO

We'll want fakeapp under apps/. Make temporary workarea and fetch working copy of apps.

cd

mkdir workarea

cd workarea

# fetch working copy of the apps directory (empty directory - no apps)

svn co $REPO --depth empty

# checkout the entire apps hierarchy

# svn co $REPO

Create directory hierarchy for our application within working copy.

cd apps

mkdir fakeapp

cd fakeapp

cp -a /localapps/runtime/fakeapp trunk

mkdir branches tags

svn add ~/workarea/apps/fakeapp

svn commit -m "initial release"

[server window] svn ls apps and see our just committed app

svn ls https://collaborate.nws.noaa.gov/svn/testsvn/ladroot/apps/

svn ls -R https://collaborate.nws.noaa.gov/svn/testsvn/ladroot/apps/fakeapp

Tag ([why tag](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions#whytag)?)

svn mkdir ~/workarea/apps/fakeapp/tags/awips1

svn cp ~/workarea/apps/fakeapp/trunk ~/workarea/apps/fakeapp/tags/awips1/1.0

svn commit -m "tagging initial awips1 version"

[user awips] -- represents a second user  
Create workarea and checkout code

cd workarea

export REPO=https://collaborate.nws.noaa.gov/svn/testsvn/ladroot/apps

svn co $REPO/fakeapp

cd fakeapp/trunk

User awips modifies fakeapp [vi fakeapp.txt to add "awips2 port mods made by user awips"]

do 'svn status -uv' from both the user awips and jho terms

* user awips will show M - working copy modified; newer vs the server
* user jho will show *(nothing) - since the server hasn't been updated*

[user awips]

svn commit -m "initial awips2 port"

do 'svn status -uv' again from both the user awips and jho terms

* user awips will show *(nothing) - working copy and server are in sync*
* user jho will show \* - working copy out of date wrt the server

[user jho]

cd trunk

cat fakeapp.txt - see original (dated) version

svn update

cat fakeapp.txt - see updated version

Tag initial awips2 version. Use [AppSvnTagRelease](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/AppSvnTagRelease) to do tagging, but here are the manual steps as an example.  
[user awips, from /home/awips/workarea/fakeapp]

svn mkdir tags/awips2

svn cp trunk tags/awips2/1.1

svn commit -m "tagging initial awips2 version"

[user jho]

svn status -uv

svn update

svn status -uv

**why tag?**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions#whytag)

Terminology - trunk vs branch vs tag (from stackoverflow.com)

* **trunk** - the main body of development, originating from the the start of the project until the present.
* **branch** - a copy of code derived from a certain point in the trunk that is used for applying major changes to the code while preserving the integrity of the code in the trunk. If the major changes work according to plan, they are usually merged back into the trunk. In open source projects, major branches that are not accepted into the trunk by the project stakeholders can become the bases for forks -- e.g., totally separate projects that share a common origin with other source code.
* **tag** - a point in time on the trunk or a branch that you wish to preserve. The two main reasons for preservation would be that either this is a major release of the software, whether alpha, beta, RC or RTM, or this is the most stable point of the software before major revisions on the trunk were applied.

**from ADAM**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions#fromADAM)

First, remember there are two repositories, one for testing and the other real

# Testing respository = testsvn

export REPOTESTHOME=https://collaborate.nws.noaa.gov/svn/testsvn/ladroot

# Local app respository = ncladt

export REPOHOME=https://collaborate.nws.noaa.gov/svn/ncladt/ladroot

ADAM will shortly have port 22 (ssh) open to LDAD's ls1. With that, the wiki's repository may be accessed from the ls1 with the ldad account by using svn's checkout and commit --username switch like so (tip from Jamie Frederick (TSA))

# use test repository

export REPO=$REPOTESTHOME/apps

# make workarea

ssh -l ldad ls1 mkdir /tmp/fakeapp

# checkout. user.name is your noaa ldap username

ssh -l ldad ls1 svn co --username user.name $REPO/fakeapp /tmp/fakeapp

# copy to ADAM

scp -r ldad@ls1:/tmp/fakeapp .

# delete working copy on ls1.

ssh -l ldad ls1 rm -rf /tmp/fakeapp

<make mods to fakeapp on ADAM>

# copy working copy to ls1

scp -r fakeapp ldad@ls1:/tmp

# check in. user.name is your noaa ldap username

ssh -l ldad ls1 "svn commit -m 'commit test from dx1-nhda' --username user.name /tmp/fakeapp"

# delete working copy on ls1.

ssh -l ldad ls1 rm -rf /tmp/fakeapp

**textdb[¶](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions" \l "textdb" \o "Link to this section)**

textdb in awips2 works as it does in awips1. See [here](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/HowToTextDb)

**subscription[¶](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions" \l "subscription" \o "Link to this section)**

In awips1, ingest events get triggered via the database. In awips2, that functionality is provided by a subscription service that resides within the ESB. Subscriptions must be registered with the system by using the subscription utility (/awips2/fxa/bin/subscription).

References:

* [triggers](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/triggers)
* [subscriptions](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/Subscriptions)

A simple example for registering a PIL trigger

subscription -o add -t ldad -p <PIL> -r ldad -f /localapps/runtime/bin/<filename> -c %TRIGGER%

* -t means trigger type. 'ldad' mean 'pil'. So this subscription will trigger on PILs.
* -p : the PIL
* -r means "runner". It's how the script will be run. "-r ldad" means the script will be exec'd out, and the PIL that triggered the script will be automatically retrieved from the textdb and put in the directory <edex\_home>/edex/data/fxa/trigger
* -f : the trigger script. Put in /localapps/runtime/bin -- /localapps will be mounted on all systems.
* -c %TRIGGER% - tells the server to pass to the trigger script the PIL responsible for the trigger

**GFE**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions#GFE)

* gfe is a perspective within cave
  + note: start gfe standalone (in its own perspective) with the switch -perspective: ./cave.sh -perspective gfe
* there is also a localization perspective. smartTools and utilities are edited with this perspective.
* baseline tools are kept in cave/etc/gfe/userPython on the workstation. These don't change. Each workstation has an identical copy. For example,

smartTools

/awips2/cave/etc/gfe/userPython/smartTools

utilities

/awips2/cave/etc/gfe/userPython/utilities

* site level tools

are stored locally in caveData,

caveData/etc/site/OAX/gfe/userPython/smartTools

and a copy is also stored on the server. Client and server copies are kept in sync.

/awips2/edex/data/utility/cave\_static/site/<site>/gfe/userPython/smartTools

* user level tools

are likewise written to caveData

caveData/etc/user/<user>/gfe/userPython/smartTools

and also stored on the server

/awips2/edex/data/utility/cave\_static/user/<user>/gfe/userPython/smartTools

**tools[¶](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions" \l "tools" \o "Link to this section)**

SmartScript has been ported to awips2, and *should* work much the same as it does in awips1. Each smartScript method is documented [here](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/SmartScriptMissingMethods). Differences encountered are listed [here](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/HowToGfePortingNotes). Note that eaMgr() and setUp() no longer exist.

* [numpy](http://numpy.scipy.org/) is used in place of Numeric

numpy is almost a drop in replacement for Numeric, but there are a few [differences](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/NumericToNumpy), as illustrated below

# awips1 (note Float and Int16)

from Numeric import \*

vals = self.fncValue[rec].astype(Float)/10

fncValue = self.fncData.createVariable('pop', Int16, ('record','ypts','xpts'))

# awips2 (note float and int16)

from numpy import \*

vals = self.fncValue[rec].astype(float)/10

fncValue = self.fncData.createVariable('pop', int16, ('record','ypts','xpts'))

* no more AFPS. Details [here](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/AFPS). Some workaround examples

AbsTime

# awips1

import AFPS

tTime = AFPS.AbsTime(t)

currentTime = AFPS.AbsTime\_current()

maxTime = AFPS.AbsTime\_maxFutureTime()

# awips2

import AbsTime

tTime = AbsTime.AbsTime(t)

currentTime = AbsTime.Current()

maxTime = AbsTime.maxFutureTime()

TimeRange

# awips1

import AFPS

tr = AFPS.TimeRange(t1, t2)

allTimes = AFPS.TimeRange\_allTimes

# awips2

import TimeRange

tr = TimeRange.TimeRange(t1, t2)

allTimes = TimeRange.allTimes()

**inits[¶](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions" \l "inits" \o "Link to this section)**

SmartInits - work in awips2 as they do in awips1. They're spawned differently (by EDEX's ESB), but from Init.py on down much is the same. siteConfig exists, as do serverConfig and localConfig, with localConfig overrides working as they do in awips1 (*not true at all sites? -- discuss*)

**layout, awips1 vs awips2**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions#layoutawips1vsawips2)

# awips1

from /awips/GFESuite/primary

etc/BASE/smartInit/Init.py

etc/BASE/siteConfig.py

etc/BASE/serverConfig.py

etc/SITE/localConfig.py

# awips2

from /awips2/edex/data/utility

edex\_static/base/smartinit/Init.py

edex\_static/site/<site>/config/gfe/siteConfig.py

edex\_static/base/config/gfe/serverConfig.py

edex\_static/site/<site>/config/gfe/localConfig.py

**testing[¶](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions" \l "testing" \o "Link to this section)**

Two sites, including your own, may be active at any given time. Activate and deactivate with this webpage

<http://localhost:8080/uEngineWeb/GfeServiceBackup/activate_site.html>

or the following [uengine](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/EdexUengineScripting) scripts.

Example: activate the site DMX

[jho@desmo cli]$ which uengine

/awips2/fxa/bin/uengine

[jho@desmo cli]$ uengine -r python <gfeActivateSite.py -s SITE DMX

# gfeActivateSite.py

from com.raytheon.uf.common.message.response import ResponseMessageGeneric

from com.raytheon.edex.plugin.gfe.config import GFESiteActivation

from java.lang import String

site = "%SITE%"

progName = "gfeActivateSite"

gfeSA = GFESiteActivation.getInstance()

# Set<String>

activeSites = gfeSA.getActiveSites()

isActive = False

for aSite in activeSites.toArray():

if site == aSite.toString():

isActive = True

break

if isActive == True:

msg = "%s is already active" % site

print msg

return ResponseMessageGeneric(String(msg))

try:

# returns nothing

gfeSA.activateSite(site)

except:

return ResponseMessageGeneric(String(site + " site activiation: failed"))

return ResponseMessageGeneric(String(site + " site activiation: successful"))

Example: deactivate site DMX

[jho@desmo cli]$ which uengine

/awips2/fxa/bin/uengine

[jho@desmo cli]$ uengine -r python <gfeDeactivateSite.py -s SITE DMX

# gfeDeactivateSite.py

from com.raytheon.uf.common.message.response import ResponseMessageGeneric

from com.raytheon.edex.plugin.gfe.config import GFESiteActivation

from java.lang import String

site = "%SITE%"

if site == "":

return ResponseMessageGeneric("no site specified")

try:

# returns void

GFESiteActivation.getInstance().deactivateSite(site)

except:

return ResponseMessageGeneric(String(site + " site deactiviation: failed"))

return ResponseMessageGeneric(String(site + " site deactiviation: successful"))

Example: which sites are active?

[jho@desmo cli]$ which uengine

/awips2/fxa/bin/uengine

[jho@desmo cli]$ uengine -r python <gfeActiveSites.py

# gfeActivateSites.py

from com.raytheon.uf.common.message.response import ResponseMessageGeneric

from com.raytheon.edex.plugin.gfe.config import GFESiteActivation

# Set<String>

activeSites = GFESiteActivation.getInstance().getActiveSites()

aSites = ""

for aSite in activeSites.toArray():

print "\t", aSite

if len(aSites) > 0:

aSites += ", "

aSites += str(aSite)

return ResponseMessageGeneric(aSites)

Example: manual smartInit (todo: clean up, dbName needs to be an arg)

uengine -r python <smartInit.py

# smartInit.py

from com.raytheon.uf.common.message.response import ResponseMessageGeneric

from com.raytheon.edex.plugin.gfe.smartinit import SmartInitFactory

from java.util import HashMap

argMap = HashMap()

argMap2 = HashMap()

# How is the dbName value string parsed? See DatabaseID.java, explained below.

# An example: "OAX\_GRID\_D2D\_RUC80\_20100129\_1300"

# '\_' is the token delimiter.

# token[0] = siteID

# token[1] should be "GRID"

# token[2] = dbType

# next token (3) = modelName. There can be more than one of these; separate

# with '\_'

# next to last token (date) and last token (time) are concatenated to form

# modelTime

argMap.put( "dbName","DMX\_GRID\_D2D\_RUC80\_20110121\_1600:1" )

argMap.put( "model","RUC80" )

argMap.put( "validTime",None )

print "model = ", argMap.get("model")

print "dbName = ", argMap.get("dbName")

# SmartInitScript

initScript = SmartInitFactory.constructInit()

# will throw an exception if no "d2d" data exists

ret = initScript.execute( argMap )

return ResponseMessageGeneric("successful")

Example: query the server for the serverConfig settings it has loaded

uengine -r python <ifpServerConfig.initskips.py

from com.raytheon.uf.common.message.response import ResponseMessageGeneric

from com.raytheon.edex.plugin.gfe.config import IFPServerConfigManager

# IFPServerConfig

ifpsc = IFPServerConfigManager.getServerConfig("OAX")

model = "RUC13"

print "model", model, "- hour : initskip"

for h in range(24):

# will print in request log

print h, ":", ifpsc.initSkip(model, h)

return ResponseMessageGeneric("hello, world")

**base/site/user**[**¶**](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/MigrationWorkSessions#basesiteuser)

* todo: update [this page](https://collaborate.nws.noaa.gov/trac/ncladt/wiki/BaseSiteUserDirctories)