

# Thunderstorm Auto Nowcasting Operations Guide

1. Hardware
2. Software
3. Environment Config
4. Data Display
5. Algorithms
6. Installation
7. Monitoring
8. Data Architecture
9. Procmap and the Auto-restarter

## Hardware

The Auto-Nowcast Environment contains a set of machines (or hosts) with specific tasks. Multiple machines are required to meet the resources required by the various algorithms in the Auto-Nowcast Environment. The number of machines in a given Auto-Nowcast Environment installation will vary depending on the number and types of algorithms running at that installation and the data available.

The minimal Auto-Nowcast Environment configuration consists of the two machines which have graphical data displays running on them: the Control Host and the TITAN Host. Other Hosts are added to the configuration to support non-graphical, analysis algorithms.

Throughout this document, the Auto-Nowcast Environment machines are referred to by the main function that they serve.

- \* Control Host
- \* TITAN Host
- \* Other hosts, for example, Colide Host, Ingest Host, etc.

In general, the hosts are configured identically with the same CPU, memory, swap space, and OS version. The host must be a high-end UNIX workstation with Linux 2.4.21 or higher installed.

The minimal recommended configuration for a nowcast host includes a 2.66 GHz processor with 2-3 Gb RAM and 146 Gb SCSI HD. Two ethernet cards are recommended on any machine which ingests a raw radar data from a UDP broadcast stream. This is required mainly to reduce network traffic problems. If radar data is being ingested via an LDM or TCP/IP, then a second card is not necessary.

## Software

The main software components of the Auto-Nowcast Environment are:

### Data Service

### Process Control

### Utilities

#### Data Service

---

Data in **MDV**, **SPDB**, **FMQ**, and **Titan Storm Track** formats are available through data servers. In addition, SPDB data is served out in a graphical format, Symprod, suitable for rendering on X-displays. The servers and clients communicate via TCP/IP protocol. This model allows a client (or application) running on one host within the Auto-Nowcast Environment to access data which resides on a different host. Try to co-locate data and algorithms which use the data on machines as much as possible to minimize overhead of data serving across hosts.

ASCII datasets are not served but are read directly from disk files by the various algorithms which use the ASCII data.

Following is a list of some of RAP's data service applications and their functions:

DataService	
Application	Function
<b>DataMapper</b>	Contains a map of data sets relative to <code>\$RAP_DATA_DIR</code> , used by <b>Sysview</b> , the system monitor
<b>DsMdvServer</b>	Server for data in <b>MDV</b> format.
<b>DsSpdbServer</b>	Server for data in <b>SPDB</b> format.
<b>DsFmqServer</b>	Server for <b>FMQ</b> messages.
<b>DsTitanServer</b>	Server for titan data.
<b>DsServerMgr</b>	Server manager handles requests from applications and starts necessary servers. (Thus <b>DsServerMgr</b> is the only server which needs to be included in a host process list.)
<b>Janitor</b>	Recurses through the data directory tree and compresses or removes data based on <code>_Janitor</code> param files located with the data.

#### Process Control

---

Process control refers to a simple and effective mechanism to keep realtime processes running in the Auto-Nowcast Environment. At the heart of the process control mechanism is the process mapper, a.k.a. PROCMAP, and a program called **auto\_restart** which reads and responds to the output of the process mapper.

PROCMAP						
Name	Instance	Host	User	Pid	Heartbeat	Uptime
Cidd	nowcast	desert	nowcast	18555	0:0:12	7,34 d
MDV_server	bdryGrid30	desert	nowcast	11114	0:0:55	21 d
MDV_server	forecast30_i	desert	nowcast	11059	0:0:57	21 d
MDV_server	forecast30_v	desert	nowcast	11073	0:0:52	21 d
MDV_server	nowcast30	desert	nowcast	11085	0:0:28	21 d
MDV_server	terrain	desert	nowcast	11457	0:0:28	21 d
MDV_server	titanGrid30	desert	nowcast	11098	0:0:10	21 d
VerifyGrid	realtime	desert	nowcast	10959	0:0:19	21 d
autoNowcast	ops30	desert	nowcast	10893	0:0:39	21 d
bdryGrid	nowcast	desert	nowcast	10943	0:0:17	21 d
didss_compre	nowcastIn	desert	nowcast	30641	0:0:52	18,1 d
didss_compre	nowcastOut	desert	nowcast	11150	0:0:49	21 d
fmq2tape	archive	desert	nowcast	18563	0:0:41	7,21 d
nowcastPostp	ops	desert	nowcast	10908	0:0:57	21 d
prod_sel	nowcast	desert	nowcast	29795	0:0:33	19 d
radarTrigger	nowcast	desert	nowcast	10866	0:0:5	21 d
ridds2nom	nowcast	desert	nowcast	10766	0:0:7	21 d
servmap	servmap	desert	nowcast	10669	0:0:42	21 d
titanGrid	nowcast	desert	nowcast	10928	0:0:5	21 d
vergrid_spdb	VerGrid	desert	nowcast	11126	0:0:17	21 d
MDV_server	radialRadar	force	nowcast	1847	0:0:16	7,2 d
bdry_spdb2sy	bdryDetect	force	nowcast	1640	0:0:16	14,3 d
bdry_spdb2sy	bdryExtrap30	force	nowcast	1099	0:0:38	14,3 d
colide	ops	force	nowcast	1187	0:0:10	14,3 d
didss_compre	radialIn	force	nowcast	1232	0:0:15	14,3 d
get_nexrd	colide	force	nowcast	1284	0:0:12	14,3 d
gint	colide	force	nowcast	1366	0:0:4	14,3 d

All of the realtime processes within the Auto-Nowcast Environment register with the process mapper on a regular interval. This regular registration, or heartbeat, provides an indicator of the status of each realtime process in the Auto-Nowcast Environment. **auto\_restart** compares the output of the process mapper with a user configured process list and makes sure all of the processes in the list are up and running and registering at proper intervals.

Following is a list of utilities, applications, and scripts used for process control:

Control Utilities, Applications and Scripts	
Utility, Application, or Script	Function
<b>cron</b>	<b>cron</b> is a UNIX utility which is a daemon used for executing commands on a timed schedule.
<b>procmap</b>	<b>procmap</b> is an application with which system applications register on a regular basis.

<b>auto_restart</b>	<b>auto_restart</b> is an application which uses <b>procmmap</b> and a process list to insure that all processes in the list are running and checking in with <b>procmmap</b> at regular intervals. If a process is missing, <b>auto_restart</b> will use the start script in the process list to restart it. If a process is hung, <b>auto_restart</b> will use the 'kill' and 'start' scripts in the process list to kill it and then restart it.
<b>procmmap_list_start</b>	<b>procmmap_list_start</b> starts all processes in a process list with the start scripts for each process indicated in the list.
<b>start_process.control</b>	<b>start_process.control</b> is a script run by <b>cron</b> every five minutes. If necessary, this script starts <b>procmmap</b> , the processes in the process list with <b>procmmap_list_start</b> and <b>auto_restart</b> .
<b>host_startup</b>	<p><b>host_startup</b> executes the start_up sequence for a single host in the system. Here is the sequence:</p> <ol style="list-style-type: none"> <li>1. Concatenate relevant <b>\$CONTROL_DIR/params/host_env.*</b> files and install in <b>\$CONTROL_LOCAL_DIR/runtime/HOST_ENV</b>. This file contains the system hosts and their roles.</li> <li>2. Install <b>\$CONTROL_DIR/params/radar_env.lookup</b> in <b>\$CONTROL_LOCAL_DIR/runtime/RADAR_ENV</b> (which is used for text substitution in process list.)</li> <li>3. Concatenate relevant process lists from <b>\$CONTROL_DIR/proc_list</b> based on the host roles in <b>HOST_ENV</b> and install in <b>\$CONTROL_LOCAL_DIR/runtime/PROCESS_LIST</b></li> <li>4. Concatenate relevant cron files from <b>\$CONTROL_DIR/crontab</b> and install in <b>\$CONTROL_LOCAL_DIR/runtime/CRONTAB_LIST</b> and start cron.</li> <li>5. Start <b>start_process.control</b> with <b>procmmap_list_start</b> and <b>auto_restart</b> using <b>\$CONTROL_LOCAL_DIR/runtime/PROCESS_LIST</b>.</li> </ol>
<b>host_shutdown</b>	<p><b>host_shutdown</b> executes the shutdown sequence for a single host in the system. Here is the sequence:</p> <ol style="list-style-type: none"> <li>1. kill <b>cron</b></li> <li>2. kill <b>auto_restart</b></li> <li>3. kill processes in the process list</li> <li>4. kill data servers</li> <li>5. kill <b>procmmap</b></li> <li>6. remove <b>FMQs</b> if part of entire system shutdown.</li> <li>7. remove <b>HOST_ENV</b></li> </ol>
<b>niwot_startup</b>	<b>niwot_startup</b> executes <b>host_startup</b> on each host in the system.
<b>niwot_shutdown</b>	<b>niwot_shutdown</b> executes <b>host_shutdown</b> on each host in the system.

## Utilities

Following is a list of some handy utilities and their functions:

Utilities
-----------

<b>Application</b>	<b>Function</b>
<b>PrintMdv</b>	Print contents of mdv file.
<b>SpdbQuery</b> or <b>XSpdbQuery</b>	Query an <b>SPDB</b> database
<b>print_procmap</b>	prints the processes currently registering with <b>procmap</b>
<b>RadMon</b>	View radar beam data as it is being written to an <b>FMQ</b>
<b>snuff</b> or <b>snuff_inst</b>	kills processes or instances of processes
<b>utime_decode</b>	decodes unix time into UTC and local time
<b>LogFilter</b>	Writes application output in dated subdirectory format. (All of the start scripts for applications pipe the application output to <b>LogFilter</b> )
<b>running</b>	Returns 0 if process is running and 1 if it is not. (Most of the start scripts use this utility before starting an application so that more than one instance of an application will not start.)

## Environment Configuration

The Project Directory: Application Param Files and Start Scripts, Host Config, and Data Config.  
 Environment Variables  
 Network Communications

### The Project Directory: Application Param Files and Start Scripts, Host Config, and Data Config.

The project directory or **\$PROJ\_DIR** contains all project specific information including: "homes" for all of your project specific application parameter files and start scripts, project host configuration and process lists for each host, cvs project checkout script and build scripts, data directories and data param files for each host, and system monitor.

Following are the three subdirectories of **\$PROJ\_DIR** that contain all host, data, and software configuration information for a particular installation of the AutoNowcaster.

<b>\$PROJ_DIR subdirectories for process control, and host data and software configuration</b>	
<b>Directory</b>	<b>Contents</b>
<b>control</b>	<b>crontab</b> -- crontab files for each host <b>params</b> -- email.list for nightly stats host_env.<mode> (host/hostRole pairs) mode.equiv (system startup mode equivalences) radar_env.lookup (radar/host pairs + identifying keyword) <b>proc_list</b> -- process lists for each host <b>runtime</b> -- some runtime files including:  NIWOT_LOG -- system startup times, shutdown times, and mode
<b>dataHome</b>	<b>bin</b> -- start and kill scripts for data service apps. <b>DsServerMgr, Janitor, DataMapper</b> <b>params</b> -- param files that will be copied to data directories (like the _Janitor files), plus static terrain files. <b>data_lists</b> -- lists of data directories on each host and param files that should reside in particular data directories.
<b>utilsHome</b>	<b>bin</b> contains checkout and build scripts of all apps and libs used in a particular installation of the autoNowcaster.

Following is a list of homes for applications. Each application home generally contains two subdirectories: **bin** and **params**. The **bin** subdir contains all start scripts and kill scripts(if there are any) for the applications. The **params** subdir contains all of the application parameter files.

<b>\$PROJ_DIR subdirectories: Homes for applications</b>		
<b>Home</b>	<b>Description</b>	<b>Applications</b>
<b>adjointHome</b>	Home to adjoint.	<b>adjoint</b>
<b>advectHome</b>	Home to all applications related to advection.	<b>ctrec</b> <b>advectGrid</b> <b>stratiform_filter</b>
<b>ciddHome</b>	Home to all applications related to CIDD the Configurable Interactive Data Display.	<b>CIDD</b> <b>DrawFmq2Bdry</b> <b>GridPointSelect</b>

		<b>Sounding2Metar</b> <b>SoundingPlot</b> <b>SoundingText</b>
<b>cronusHome</b>	Home to all applications which generate data for the forecast (with the exception of titanGrid which is in <b>titanHome</b> ).	<b>bdryCollision</b> <b>bdryGrid</b> <b>bdryStormCol</b> <b>cronus</b> <b>gandi</b> <b>radarTrigger</b>
<b>ingestHome</b>	Home to all applications which ingest data into the AutoNowcast environment (except radar ingest which has its own home.)	<b>LdmDynamic2Static</b> <b>Metar2Spdb</b> <b>NidsVad2Spdb</b> <b>derived_ruc_fields</b> <b>kavltg2spdb</b> <b>ruc_sounding</b> <b>surf_interp</b>
<b>radarHome</b>	Home to all radar ingest, quality control, and reformatting applications.	<b>nexrad2dsr</b> <b>Dsr2Vol</b> <b>MdvMerge</b> <b>BrightBand</b> <b>RadMon</b> <b>vadAnalysis</b>
<b>titanHome</b>	Home to Titan and Titan display applications	<b>Titan</b> <b>Rview</b> <b>TimeHist</b> <b>Tstorms2Spdb</b>
<b>sysviewHome</b>	Home to applications, params files, and diagrams related to the system monitor called SysView.	<b>SysView</b>

## Environment Variables

---

The Auto-Nowcast Environment is run under the special user account nowcast. The nowcast user account makes heavy use of UNIX environment variables for navigating around the Auto-Nowcast Environment. Environment variables provide an easy mechanism for installing a generic Auto-Nowcast Environment setup at different field sites. \$PROJ\_DIR subdirectories for process control, and host, data and software configuration.

### Rap directory

\$RAP\_DIR /nfs/ncar/rap

### Bins, Libs, and Includes Environment Variables

\$RAP\_SHARED\_BIN\_DIR /nfs/ncar/bin

\$RAP\_SHARED\_INC\_DIR /nfs/ncar/inc



\$RAP\_SHARED\_LIB\_DIR /nfs/ncar/lib  
\$RAP\_BIN\_DIR /nfs/ncar/rap/bin  
\$RAP\_INC\_DIR /nfs/ncar/rap/include  
\$RAP\_LIB\_DIR /nfs/ncar/rap/lib

### **The Project Directory Environment Variables**

\$PROJ\_DIR \$RAP\_DIR/<project name>  
\$ADJOINT\_HOME \$PROJ\_DIR/adjointHome  
\$ADVECT\_HOME \$PROJ\_DIR/advectHome  
\$CONTROL\_DIR \$PROJ\_DIR/control  
\$CIDD\_HOME \$PROJ\_DIR/ciddHome  
\$COLIDE\_HOME \$PROJ\_DIR/colideHome  
\$CRONUS\_HOME \$PROJ\_DIR/cronusHome  
\$DATA\_HOME \$PROJ\_DIR/dataHome  
\$CRONUS\_HOME \$PROJ\_DIR/cronusHome  
\$COLIDE\_HOME \$PROJ\_DIR/colideHome  
\$RADAR\_HOME \$PROJ\_DIR/radarHome  
\$SATELLITE\_HOME \$PROJ\_DIR/satelliteHome  
\$SYSVIEW\_HOME \$PROJ\_DIR/sysviewHome  
\$TITAN\_HOME \$PROJ\_DIR/titanHome  
\$UTILS\_HOME \$HOME/utilsHome

### **Data and Logs Environment Variables**

\$RAP\_DATA\_DIR \$HOME/data  
\$LOG\_DIR \$HOME/logs

### **Network Communications**

---

#### **Cross Mounted File Partition**

All the hosts in the Auto-Nowcast Environment cross-mount (via NFS) one file partition called \$RAP\_SHARED\_DIR. The partition is exported with read/write permissions to all Auto-Nowcast Environment hosts. \$RAP\_SHARED\_DIR contains the NCAR executable applications as well as the project directory (\$PROJ\_DIR) in which the components of Auto-Nowcast Environment resides.

#### **Local Area Network**

The hosts within the Auto-Nowcast Environment communicate using TCP/IP protocol.

#### **Radar Network**

Each Auto-Nowcast Environment host which has radar data coming into it from a UDP broadcast has a second ethernet card for that purpose. The use of a second ethernet card for broadcasting radar data protects the load on the local area network in with the Auto-Nowcast Environment is installed. If radar data is being broadcast via TCP/IP there is no need for a second ethernet card.

## Data Display

Two types of graphical data displays are available in the Auto-Nowcast Environment:

[Cidd](#)

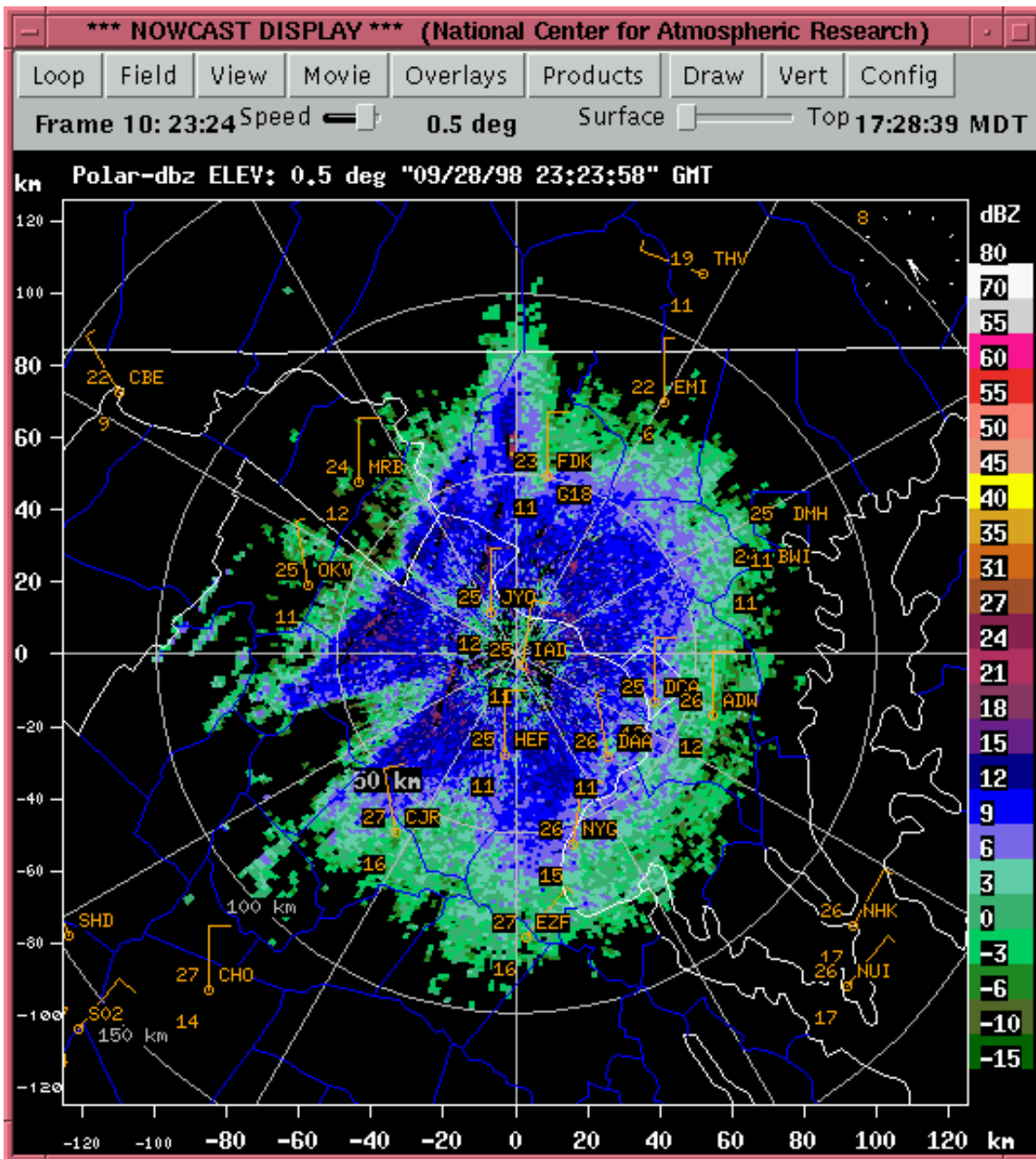
[Rview](#)

### **Cidd**

---

Is the most general of the data display applications. It is highly user-configurable and is capable of displaying both MDV and SYMPROD (the graphical form of SPDB) data.

Cidd is a X-based, real-time data display application which supports PPI, CAPPI, and cross sectional views. Cidd is the main display tool within the Auto-Nowcast Environment. It is highly configurable and is used to display the majority of the MDV and SPDB formatted data produced within the Environment.

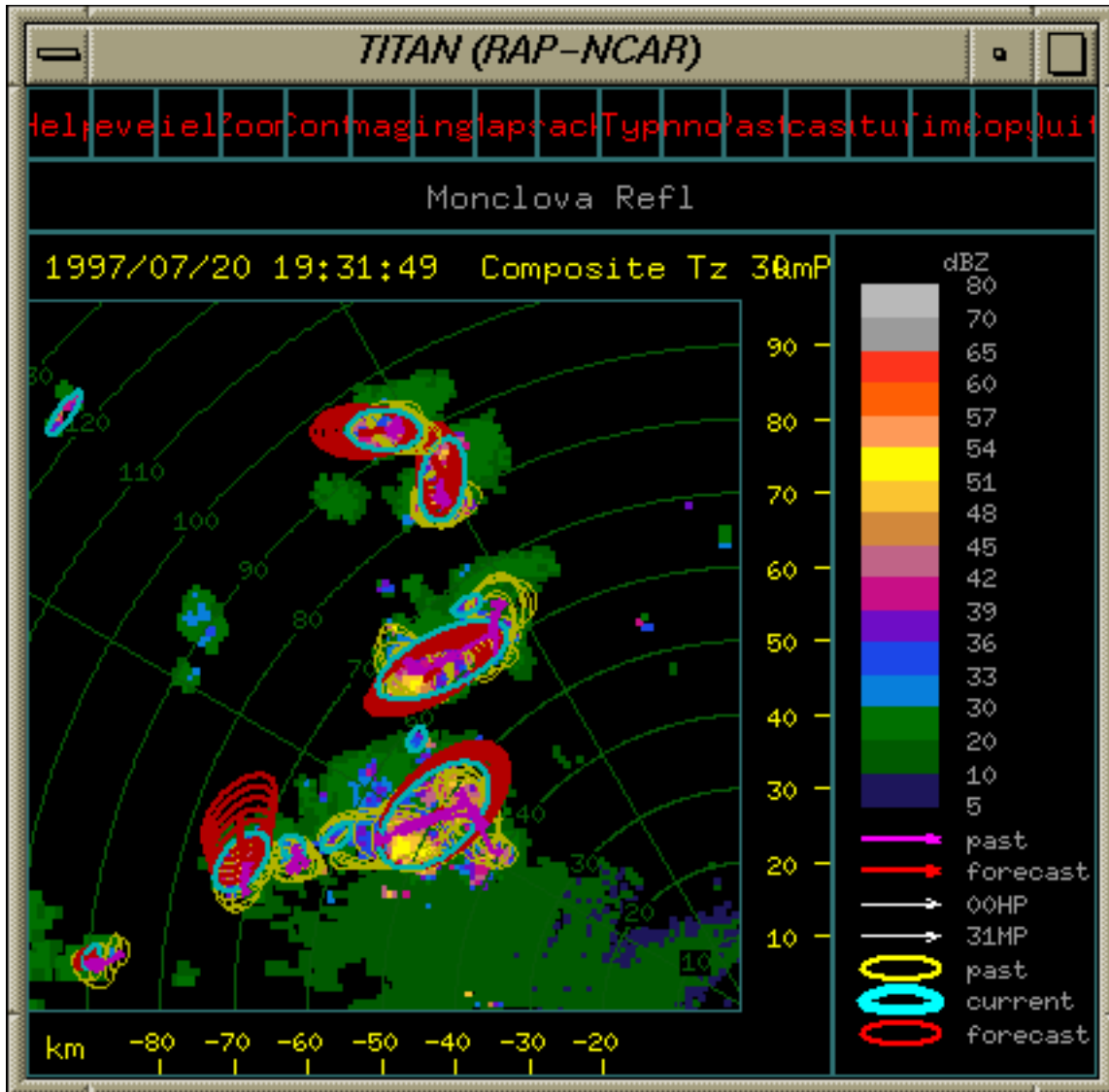


### Rview

Is part of the TITAN application and provides a detailed graphical view of the thunderstorm tracking algorithm.

The X-based display utility Rview is part of the TITAN analysis algorithm and is used to display thunderstorm tracks in real-time over top of CAPPI radar data as well as vertical sections through storms.

For more information on the TITAN application and using the Rview display, see TITAN: Thunderstorm identification, tracking, analysis and nowcasting - a radar-based methodology and The TITAN User's Manual.



# Analysis Algorithms

- The Nowcast Engine
- Thunderstorm Identification, Tracking, and Features
- Advection
- Variational Doppler Radar Analysis
- Satellite Algorithms
- Data Ingest and Quality Control

## The Nowcast Engine

---

### Cronus

Cronus gathers and combines the relevant user defined meteorological data that have been generated by the various algorithms with user defined functions and weights to produce an initiation interest field and a growth and decay interest field. These These two interest fields as input into gandi.

Cronus is triggered to run by radarTrigger which watches the incoming radar beam data and sends a message or 'trigger' to cronus when a user specified tilt has begun. Cronus then sends a forecast trigger to algorithms producing data used to create the interest fields indicating that the algorithm should run and produce forecast data for specific time in future. Cronus then gathers output from the algorithms and produces its interest fields.

### Gandi

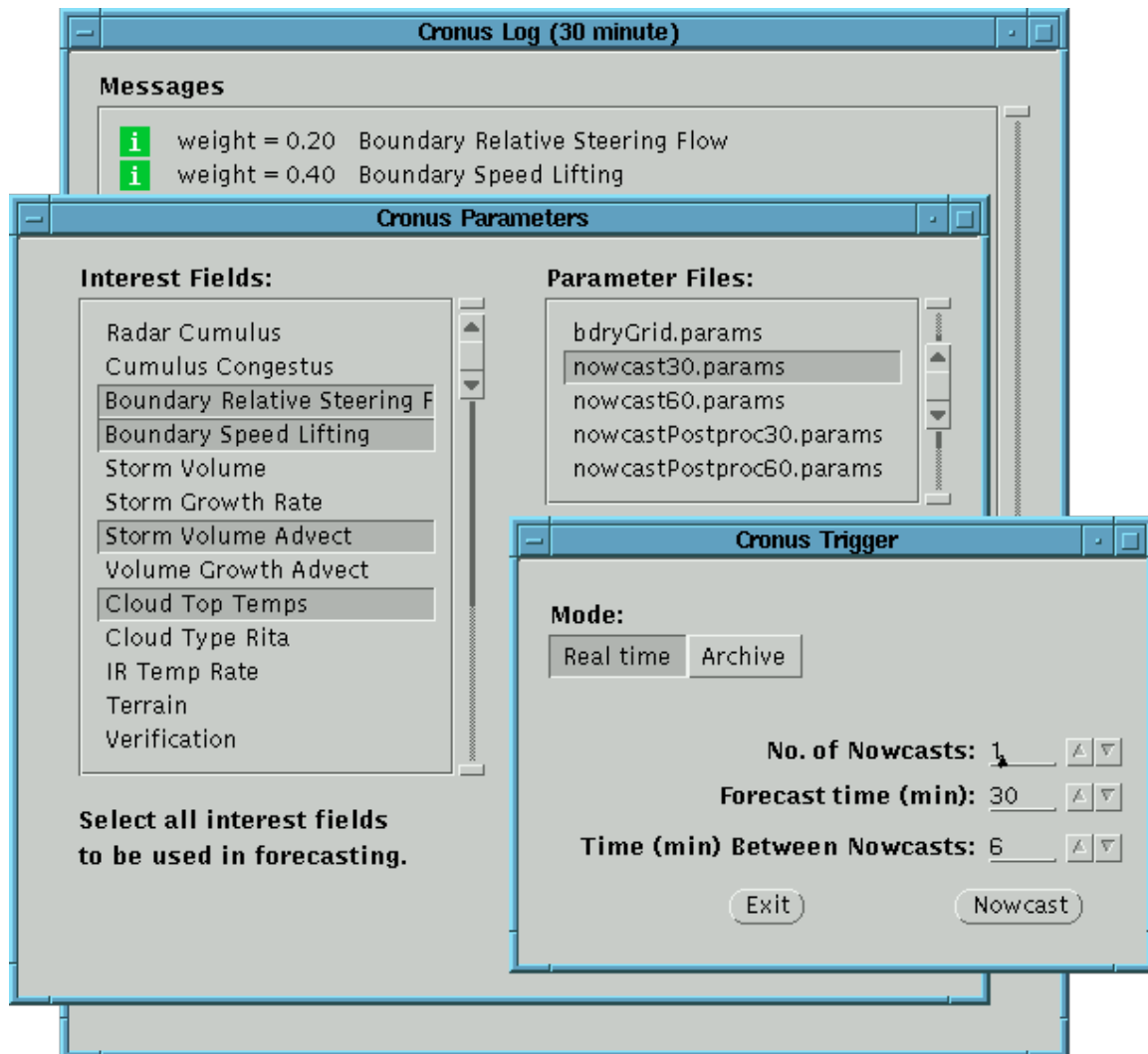
Gandi takes as input the initiation and growth and decay interest fields which are output by cronus and the dbz advect field. User defined step functions are applied resulting in a final reflectivity or precip rate forecast. The initiation interest field is used to calculate and initiation reflectivity field (based on user defined step function.) The growth and decay interest field is used to modify the dbz advect field, either growing or decaying regions above a user specified reflectivity threshold. Finally, the initiation and growth and decay reflectivity fields are merged.

**Scripts:**                   \$CRONUS\_HOME/bin  
**Parameters:**           \$CRONUS\_HOME/params

Executables	Input Data		Output Data	
	Description	Format	Description	Format
radarTrigger	radar tilt indicator	fmq	tilt trigger	fmq
	tilt trigger	fmq	forecast trigger	fmq
cronus	user selected interest fields	MDV	initiation or growth and decay interest fields	MDV
gandi	advected reflectivity field	MDV	2D reflectivity field	MDV
	initiation or growth and decay	MDV		

interest fields

Cronus operates in realtime mode by triggering at the start of the 0.5 degree radar tilt. When the nowcast domain includes several radars, the trigger can be set by MdvMerge , the application which merges the cartesian radar data from all radars. In addition to the realtime trigger, the user can manually trigger a nowcast at anytime using the graphical interface to the cronus application.



## Thunderstorm Identification, Tracking, and Features

TITAN is a collection of processes which identify and track thunderstorm above a user-specified reflectivity (dBZ) threshold.

Rvview and TimeHist are both graphical [data displays](#) for Titan output.

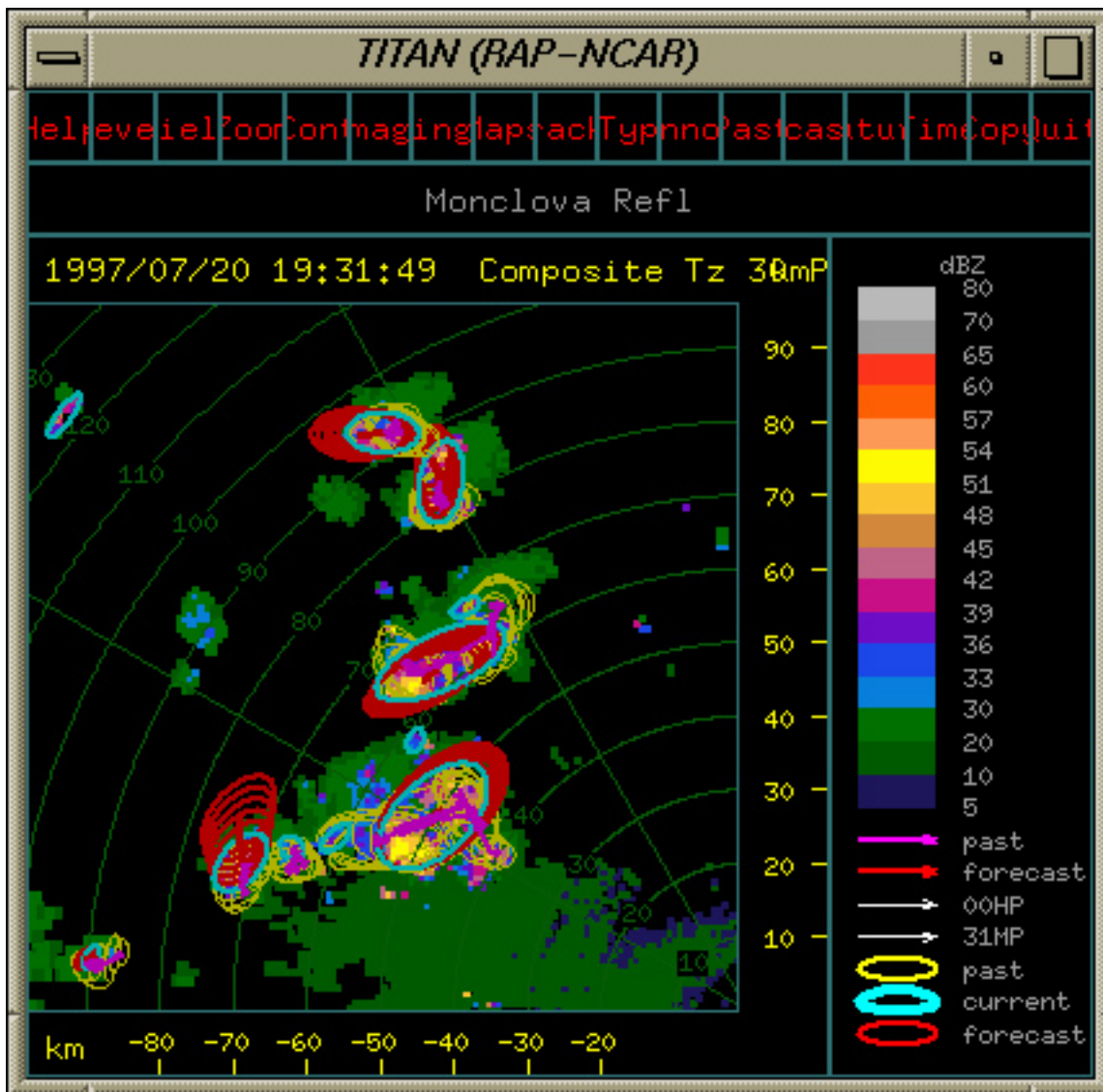
titanGrid converts titan storms characteristics into gridded fields. It can also tag storms with point statistics such as lightning strike statistics and probability of hail. Optionally, storms can be extrapolated using titan motion information.

StormInitDetect detects new storms from Titan and outputs the latitude, longitude, and time of storm initiations.

StormInit2Field grids the output of StormInitDetect and produces regions of storm initiation defined by gaussians at each storm initiation location.

**Scripts:**                    \$TITAN\_HOME/bin  
**Parameters:**                \$TITAN\_HOME/params

Executables	Input Data		Output Data	
	Description	Format	Description	Format
Titan	cartesian radar data	MDV	thunderstorm information	Titan StormTrack
titanGrid	Titan thunderstorm information	Titan Storm Track	2D gridded storms tagged with storm characteristics or point data statistics	MDV
	point data	SPDB		
StormInitDetect	Titan thunderstorm information	Titan Storm Tracks	Points of thunderstorm initiation	SPDB
StormInit2Field	Points of thunderstorm initiation	SPDB	Gridded regions of thunderstorm initiation	MDV
Rview	Cartesian reflectivity data	MDV	Graphical data display	
	Storm information	Titan Storm Tracks		
TimeHist	Storm information	Titan Storm Tracks	Graphical data display	



For more information and detail on the TITAN application, see [TITAN: Thunderstorm identification, tracking, analysis and nowcasting – A radar-based methodology](#) and the *TITAN User's Manual*.

### Advection

ctrec employs cross-correlation analysis of echoes to calculate motion vectors from radar data. ('trec' is an acronym for Tracking of Radar Echoes by Correlation.)

advectGrid uses ctrec motion vectors or sounding data to advect gridded data. It is usually used to advect reflectivity and storm characteristics.

startiform\_filter filters flat echoes from radar data at user specified elevation. Used to isolate radar cu.



**Scripts:** \$ADVECT\_HOME/bin  
**Parameters:** \$ADVECT\_HOME/params

Executables	Input Data		Output Data	
	Description	Format	Description	Format
ctrec	cartesian radar data	MDV	motion vector field	MDV
advectGrid	motion vector field from ctrec	MDV	2D advected dataset image	MDV
	sounding data used as backup to motion vector field	SPDB		
	dataset to be advected	MDV		
stratiform_filter	cartesian radar data	2D filtered dataset	MDV	

For more information on ptrece see [Determination of the Boundary Layer Airflow from a Single Doppler Radar](#).

### **Variational Doppler Radar Analysis (VDRAS)**

---

VDRAS is a system that produces high-resolution three-dimensional wind in the boundary layer. The wind field is obtained through a retrieval process combining single-Doppler radar observations and a numerical model using a four-dimensional variational scheme. The output fields from VDRAS include three velocity components, divergence, and temperature.

**Scripts:** \$ADJOINT\_HOME/bin  
**Parameters:** \$ADJOINT\_HOME/params

Executables	Input Data		Output Data	
	Description	Format	Description	Format
adjoint	PPI radar volume	MDV	3D wind, divergence, and temperature fields	MDV
	surface observations	SPDB		
	soundings	SPDB		

For more information on the adjoint method see [Forecasting Storm Growth and Decay using Low-level Radar Data and the Adjoint Method](#).

### **Satellite Algorithms**

---

satDerive calculates differences and standard deviations of satellite channel data. Calculates short wave, infrared reflectance, an icing index.

satThresh thresholds satellite data. Threshold values are applied to the visible and clear IR channels. Then these fields are used to mask all other fields, so that only pixels at locations that have passed the thresholding tests on IR and the visible are output - the rest are marked as bad or missing.

RateOfChange calculates the rate of change of MDV gridded data. Detects cloud growth by monitoring cloud top temperatures from infrared satellite data.

CloudClass classifies clouds based on a set of threshold based rules. Thresholds can vary sinusoidally according to the time of the day or year.

**Scripts:**                \$SATELLITE\_HOME/bin  
**Parameters:**        \$SATELLITE\_HOME/params

Executables	Input Data		Output Data	
	Description	Format	Description	Format
satDerive	raw satellite data	MDV	derived satellite data fields	MDV
satThresh	satellite data	MDV	thresholded satellite data	MDV
RateOfChange	IR Satellite Data	MDV	2D rate of change data values	MDV
CloudClass	satellite data	MDV	bitwise map of cloud type	MDV

### Data Ingest and Quality Control

The algorithms in the Auto-Nowcast Environment operate on data RAL's internal data formats: **FMQ, MDV, SPBD, or Titan Storm Tracks**. Many data ingesters/converters have already been written and are readily available. For other datasets, special converters may need to be developed to bring the data stream into the Auto-Nowcast Environment. Following is a list of data ingest and quality control applications:

Application	Description	Input	Output
<b>nexrad2dsr</b>	Level-2 NEXRAD radar data ingester	Level-2 NEXRAD from LDM, TCP/IP port	DsRadar written to FMQ
<b>JamesD</b>	Dealises radar data	DsRadar	DsRadar written to FMQ
<b>Dsr2Vol</b>	Collects radar beam data and writes radar volumes in various coordinates (polar, ppi, cartesian.)	DsRadar from FMQ	DsRadar written to FMQ
<b>BrightBand</b>	Removal of enhanced reflectivity in melting layer due to hydrometeor aggregation and water coating.	MDV (cartesian coordinates)	MDV (cartesian coordinates)

<b>ApFilter</b>	Anomalous propagation filter	MDV (cartesian coordinates)	MDV (cartesian coordinates)
<b>NidsVad2Spdb</b>	Ingests and reformats NEXRAD Level 3 Vad data	NIDS VAD data	Spdb
<b>Terrascan2Mdv</b>	Ingests satellite data	Terrascan	MDV
<b>Metar2Spdb</b>	Ingests and reformats surface observations	METAR	Spdb
<b>sndgIngest</b> or <b>classIngest</b>	ingests and reformats sounding data	class	Spdb
<b>NWSsoundingIngest</b>	ingests and reformats sounding data	WMO	Spdb
<b>kavltg2spdb</b>	ingest and reformats lightning data	Kavaouras	Spdb
<b>rucIngest</b>	ingest RUC model data	RUC GRIB	MDV
<b>derived_ruc_fields</b>	derive additional fields from primary RUC fields	MDV	MDV
<b>ruc_sounding</b>	Create a sounding from RUC model data	MDV	SPBD

## Installation

- Using This Configuration Guide
- Hardware Requirements & Purchasing
- UNIX System Setup
- The Nowcast User Account
- Shared Project Space
- Creating a new Auto-Nowcast Project in CVS
- Checking out an Existing Auto-Nowcast Project from CVS
- Modifying & Installing .dot Files
- Establishing ssh authentication
- Modifying Process Control Files
- Defining the Data Hierarchy
- Building Libraries & Applications
- Modifying Parameter Files
- Setting up a System Monitor
- Ready, Set, Go...
- Freezing the Source Code
- Packaging up the Field Project
- At the Field Site

### Using This Configuration Guide

---

The Auto-Nowcast system has been installed more than a dozen times over the last 8 years using varying system layouts, configuration techniques, scripts, and algorithms. By and large the configuration has undergone significant changes with each installation, always striving for improvements over the previous installation. Keeping in mind that every approach has had its share of strengths and weaknesses, the following should be considered a **guide** not a cookbook recipe. To ease the burden of the installation process, explicit command are often included; however, these should be interpreted more as suggestions than instructions. This guide is neither complete nor comprehensive, but reflects a suggested configuration largely based on the Thor2003 project.

The UNIX commands in this installation guide are expressed in a combination of **font style** and the international standard EBNF notation for language syntax including:

- |  
meaning "or"
- <>  
angle brackets used to surround non-terminal symbols
- []  
square brackets used to surround optional items
- { }  
curly brackets used to surround repetitive items of zero or more
- ()  
parentheses used for grouping

The angle brackets are to distinguish non-specific syntax (also known as non-terminal symbols) from terminal symbols which are written exactly as they are to be executed. For example, the command:

```
mv projects/<prototypeProject> projects/<newProjectName>
```

indicates that the reader is to substitute exact content for the category names `prototypeProject` and `newProjectName` while all other parts of the command are to be types as is. The reader might therefore execute the command:

## Hardware Requirements & Purchasing

---

Before purchasing hardware for the Auto-Nowcast environment, you must determine which algorithms will be part of the field project and which machines will play which roles within the project. This determination will dictate how many machines should be purchased.

Typical Auto-Nowcast Machines	
Role	Logical Name(s)
Process Control & Graphical Display	\$CONTROL_HOST \$DISPLAY_HOST
Storm Identification & Tracking	\$TITAN_HOST
Data Ingest	\$INGEST_HOST
Boundary Detection	\$COLIDE_HOST
Satellite Applications	\$SATELLITE_HOST
Ctrec Advection	\$ADVECT_HOST
Winds Modelling	\$ADJOINT_HOST
Final Nowcast	\$CRONUS_HOST
Field Display	\$FIELD_DISPLAY_HOST

The current hardware recommendations for each of the machines in the Auto-Nowcast environment include:

- 2.66 GHz dual processors
- 3 Gb RAM
- One or two 146Gb HD

## UNIX System Setup

---

The first step in setting up the Auto-Nowcast machines is to decide upon a file partitioning. A reasonable recommendation is 2Gb for the root partition, 1Gb for the /home partition, with everything else in a data partition, say /d1.

Later, in the section [Shared Project Space](#), when it is necessary to establish areas to be used for cross mounts, suggestions are made based on the existence of a /d1 partition.

The next step in the UNIX system setup for the Auto-Nowcast environment is to install the following required software:

- Debian Linux version 3.0 or greater
- gcc/g++/egcc (libc6 required)

- g77
- gdb
- ddd
- pthread library
- ssh
- NTP (network time protocol) service
- The perl and python interpreters
- The mysql libs (for Metar2Spdb)

Other support software which may need to be installed includes:

- Xview on the machine(s) which will compile and/or run CIDD
- ImageMagick on the \$DISPLAY\_HOST
- Java if we are running SysView

## **The Nowcast User Account**

---

Create the following user account on all project machines:

User: nowcast

UID : 10140

GID : 678 (shared)

There are two types of passwords that are used for the nowcast user account.

1. The internal password is used for all machines in RAP which have a nowcast account. This password does not change.
2. The external password is used on Auto-Nowcast project machine which are out in the field. This password is project-specific, determined by the lead engineer on the project, and can be changed.

While project machines are inside of RAP being set up for a field project, they should use the internal password. Later, [at the Field Site](#), you will change over to the external password.

Make sure that all other project members who get accounts on the field project machines are setup with the group "shared" (GID 678) as their default group. This is necessary for interactions with CVS.

## **Shared Project Space**

---

The Auto-Nowcast project space contains source code, libraries, application executables, and the field project configuration. At RAP these components are contained on the cross mount **\$RAP\_SHARED\_DIR** (/rap) and **\$RAP\_DIR** (usually ~/rap). For Auto-Nowcast installation we do not use /rap in order to avoid confusion and in order to allow the Auto-Nowcast user nowcast to do a **make\_install\_shared** into a project-specific location rather than installing executables into /rap/bin. This approach allows us to take a snap-shot of the software libraries and applications that will be fielded. As is done at RAP, the **\$RAP\_SHARED\_DIR** is installed on a cross mounted partition which we call the shared project space. To begin installing an Auto-Nowcast environment, you must select which machine you will use for serving the shared project space. Plan on at least 2 Gb to be used for the shared project space. It is recommended that the **\$RAP\_SHARED\_DIR** is cross mounted in the following way:

On **<Shared Project Space Host>**, the machine where the disk is physically mounted:

1. Login as root
2. Create **/d1/ncar** writable by group **shared**, owned by user **nowcast** (Use unix commands **chown** and **chmod**)
3. Create **/nfs**
4. Create a link from **/nfs/ncar** to **/d1/ncar**
5. Add the following entry to **/etc/exports**: **/d1/ncar <projectHostNames>(rw)**
6. Execute the following commands:  
**/etc/init.d/nfs-kernel-server stop**  
**/etc/init.d/nfs-kernel-server start**

On all other project machines:

1. Login as root
2. Create **/nfs**
3. Add the following entry to **/etc/auto.mnt**: **ncar <Shared Project Space host>: /d1/ncar**
4. Execute the following commands:  
**/etc/init.d/autofs stop**  
**/etc/init.d/autofs start**
5. Verify that the cross mount exists using the command **ls /var/autofs/mnt/ncar**
6. Create a link from **/nfs/ncar** to **/var/autofs/mnt/ncar**
7. Verify that the link is okay using the command **ls /nfs/ncar**

## **Creating a new Auto-Nowcast Project in CVS**

---

The next step in creating and installing a new Auto-Nowcast project is to check out an existing Auto-Nowcast project from CVS to be used as a prototype. If, however, you are simply modifying an Auto-Nowcast project which already exists in CVS, do not perform this step! Instead you should skip to the step below Checking out an Existing Auto-Nowcast Project from CVS.

**CAUTION!** The **find** command in the sequence of steps below will remove all records of CVS for the prototype project. This is an important step in setting up a new Auto-Nowcast project from a prototype project. It is important, however, that this step not be taken on an existing Auto-Nowcast project which is already checked into CVS!

1. Login as user **nowcast** on the **\$CONTROL\_HOST**
2. **cd /nfs/ncar**
3. **mkdir rapShared**
4. **cd rapShared**
5. **setenv CVSROOT :pserver:nowcast@cvs:/cvs**
6. **cvs login**
7. **cvs co utilities/niwot**
8. **cd /nfs/ncar**
9. **mkdir rap**
10. **cd rap**
11. **cvs co projects/<prototypeProject>**
12. **cd projects/<prototypeProject>**
13. **find . -name CVS -exec rm -r {} \;**
14. **cd /nfs/ncar/rap**

15. `mv projects/<prototypeProject> projects/<newProjectName>`
16. `cd projects/<newProjectName>`
17. `cvs import projects/<newProjectName> v1 r1`

Verify that the import command has read the entire tree into CVS. There have been incidents in which not all of the new project has been entered into the CVS repository by the import command.

### **Checking out an Existing Auto-Nowcast Project from CVS**

---

If you are working on an already existing Auto-Nowcast project, you must first check the project out from CVS. To do this, perform the following steps:

1. `cd /nfs/ncar/rapShared`
2. `setenv CVSROOT :pserver:nowcast@nut:/cvs`
3. `cvs login`
4. `cvs co utilities/niwot`
5. `cd /nfs/ncar/rap`
6. `cvs co projects/<projectName>`

### **Modifying & Installing .dot Files**

---

Now that you have a prototype project, the next step is to modify project-specific files to reflect the correct settings for the new Auto-Nowcast project. The first set of files which need project-specific modifications are a subset of the .dot files which reside in `$RAP_DIR/$PROJECT` or `$PROJ_DIR`. The .dot files include:

- `.cshrc`
- `.login`
- `.nexrc`
- `.xserverrc`
- `.xtrc`
- `.xearthmarker`
- `.xinitrc`
- `.xmodmaprc`
- `.fvwm`

After making necessary modifications to some of these files, we will set up links to these files in `$HOME` for user `nowcast`.

As user `nowcast` on the `$CONTROL_HOST`

1. `cd`
2. `rm .cshrc -or- mv .cshrc .cshrc.bak`
3. `ln -s /nfs/ncar/rap/projects/<newProjectName>/.cshrc`
4. Edit the file `.cshrc` and change the environment variable for `$PROJECT` to reflect the new project name, i.e., `setenv PROJECT <newProjectName>`
5. `source .cshrc`
6. `cd $PROJ_DIR`



7. Edit the files in the **.fvwm.\*** directories. These files configure the window manager on various hosts based on the host function. A minimal configuration of the window manager might be designed like this: From the **\$CONTROL\_HOST**, the pull down menus include remote logins to all nowcast hosts, and system startup and shutdown. From the **\$DISPLAY\_HOST**, pull down menus are created to start and stop CIDD( the nowcast display) and login to hosts relevant from the display site. From all other hosts, menus include only remote logins.

□ In **.fvwm.\***:

- **init.hook**: In the command that execs **xearth**, put in the latitude and longitude of the vantage point from which you wish to view the earth. (Usually this is midway between the field site and NCAR.)
- **main-menu.hook**: Change machine names under **Remote\_Logins**. Add or delete additional pull-down menu items for project.
- **post-hook**: Edit "NIWOT SYSTEM" menu items to be consistent with main-menu.hook menus.

□ In **.fvwm.control only**:

- **main-menu.hook**: Add or delete modes for system startup.

8. Edit the file **.xearthMarker** to include the lat/lon position of the field project.

To install the new **.dot** files on each of the hosts, we will use the script **setupNowcastUser.home**. In addition to installing the **.dot** files, this script will create

directories **\$HOME/tmp**, **\$HOME/controlLocal/runtime** which is the directory which will contain the [runtime process control files](#). The different options **-c** and **-f** for **setupNowcastUser.home** result in different window manager configurations based on host function.

As user **nowcast** on the **\$CONTROL\_HOST**

1. **setupNowcastUser.home -c -v -p \$PROJ\_DIR**

As user **nowcast** on the **\$FIELD\_DISPLAY\_HOST**

1. **In -s /nfs/ncar/rap/projects/<\$PROJECT>/.****csorc**

2. **source ~/.csorc**

3. **setupNowcastUser.home -f -v -p \$PROJ\_DIR**

As user **nowcast** on all remaining hosts

1. **In -s /nfs/ncar/rap/projects/<\$PROJECT>/.****csorc**

2. **source ~/.csorc**

3. **setupNowcastUser.home -v -p \$PROJ\_DIR**

## **ssh Authentication**

---

The next step is to establish **ssh** authentication between all of the project machines. Because many of the Auto-Nowcast process control scripts rely on **ssh**, it is necessary to establish **ssh** logins which do not involve password prompting for user **nowcast**.

To set up **ssh** authentication you need to create two files: **known\_hosts** and **authorized\_hosts** and distribute these files to each machine within the Auto-Nowcast system. Start by picking one host, we'll call this **<STARTING\_HOST>**:

1. Login to the **<STARTING\_HOST>** host and create/edit the file **\$HOME/authorized\_keys.tmp** We will generate host keys on each host and copy them to this temporary working file.

2. In another xterm, login to each host sequentially (including the **<STARTING\_HOST>**) and do the following:

- If asked, answer **yes** to the question:

**Are you sure you want to continue connecting (yes/no)?**

- Once you are logged on, run the command:

**ssh-keygen -t dsa**

(Use default filename and no passphrase.) This command will create the file **\$HOME/.ssh/id\_dsa.pub**

- **cat \$HOME/.ssh/id\_dsa.pub**
- copy and paste the output to the temporary file **authorized\_keys.tmp**  
For example, after running **ssh-keygen** on machine **gauss** the **cat** results will look like:  
**ssh-dss AAAAB3NzaC1kc3<...a long string of numbers and letters>== cnc@gauss**
- Logout of the remote host so that you are back at the **<STARTING\_HOST>**.
- In the temporary file **authorized\_keys.tmp**, make another copy of the host key, modifying the machine name to be a fully qualified domain name. For example,  
**ssh-dss AAAAB3NzaC1kc3<...a long string of numbers and letters>== cnc@gauss.rap.ucar.edu**
- 3. After completing the above steps for each machine, including the **<STARTING\_HOST>**, set the proper file mode on the temporary **authorized\_keys.tmp** file and copy it to all machines, including the **<STARTING\_HOST>**:
  - **chmod 600 \$HOME/authorized\_keys.tmp**
  - **scp \$HOME/authorized\_keys.tmp <machineName>:\$HOME/.ssh/authorized\_keys**
- 4. Now that you have logged onto each machine from the **<STARTING\_HOST>**, the **<STARTING\_HOST>** should have a file **\$HOME/.ssh/known\_hosts** which contains host ids for each machine in the autoNowcast system.
- 5. Edit this file, making a copy of each line, and modifying the machine name to be a fully qualified domain name.
- 6. Finally, copy the modified **known\_hosts** file to all other machines:
- **scp \$HOME/.ssh/known\_hosts <machineName>:\$HOME/.ssh/known\_hosts**

If password authentication is established correctly, you now should be able to **ssh** from machine to machine without password prompting. **Be sure to verify that this is the case** or you will have problems later with process control which will be very difficult to diagnose.

Now that each of the project machines has its **\$HOME** directory and cross mounts setup, you should be able to access the shared project space from any machine to make modifications to the project configuration. Note that there are aliases in the installed **.cshrc** file to facilitate moving around within the Auto-Nowcast configuration.

### **Modifying Process Control Files**

---

The next set of files which need project-specific modifications are the process control files which will eventually be installed in **\$HOME/controlLocal/runtime** by the **host\_startup** script and used for starting up and shutting down the Auto-Nowcast environment. As user **nowcast** on any host in the system, perform the following steps:

1. **cd \$CONTROL\_DIR/params**
2. Edit the file **host\_env.realtime** and change the machine names to reflect the roles that were determined in [Hardware Requirements & Purchasing](#).
3. **cd \$CONTROL\_DIR/proc\_list**
4. Edit all of the files in this directory to indicate which processes should be run on each of the logical hosts. Each entry in a proc list must contain: application name, instance name, start script, kill script, and host on which the application runs. In general the start scripts and kill scripts (other than the general utilities **snuoff** or **snuoff\_inst**) reside in one of the **\$PROJ\_DIR/\*Home/bin** directories.
5. **cd \$CONTROL\_DIR/crontab**
6. Edit all of the files in this directory to indicate which cron jobs should be run on each of the logical hosts.

## Defining the Data Hierarchy

Now you must establish the data directory tree on each host. The required data directories will depend on the algorithms which will be running operationally for the field project. As user **nowcast** on any host in the system, perform the following steps:

1. **cd \$DATA\_HOME/data\_lists**
2. Modify the files in this directory as necessary to specify the output datasets on each host.

The format of each entry in the the data list file is as follows:

**<dataDirectoryPath> local | (link <linkDestination>) { <dataDirectoryParameterFile> }**

Lines starting with a '#' are comment lines. These lines as well as blank lines are ingnored.

The **<dataDirectoryPath>** is the directory to be made. It specified relative to **\$RAP\_DATA\_DIR** if the directory path does not begin with a '/'. Directory paths beginning with '/' are treated as absolute rather than being relative to **\$RAP\_DATA\_DIR**.

The second entry is either local, indicating that the directory is local and a **mkdir -p \$RAP\_DATA\_DIR/<dataDirectoryPath>** is to be executed, or **link** indicating that the directory is a linked to another directory using the **ln -s <linkDestination> <dataDirectoryPath>** command. **NOTE!!(bug needs a fix):** The **<linkDestination>** needs to be an absolute path.

The optional series of **<dataDirectoryParameterFile>** specifies any run-time parameter files from **\$DATA\_HOME/params** to be installed into the specified data directory. This presently includes static files like terrain data files as well as param files from data managing applications. **WORK TO BE DONE!:** The static Mdv terrain data files should be moved to a different directory than **\$DATA\_HOME/params** since they arent paramter files and the script which installs them should look for them in a "static\_files" directory.

**NOTE:** These specific parameter files are for applications that operate on or manage data and are specific to the data set therefore they reside with the data. The naming convention for for the parameter file is **\_**<Data Managing Application>**.<function or identifying info>+<optional identifying information>** Note that the string following **+** will not appear as part of the filename when it is copied to the data directory. It merely allows the user to distinguish one param file from another in **\$DATA\_HOME/params** when the application is then same. Following are examples of applications that manage or operate on data and and corresponding param files:

Applications that Manage or Operate on Data and Example Parameter Files		
Application	Example Parameter Files Which Reside With the Data	Role of Application
Bdry2Symprod	_Bdry2Symprod.bdryDetect	Converts Spdb boundary products into format which can be rendered by CIDD
Tstorms2Symprod	_Tstorms2Symprod.35dbzStormsDetect	Converts Spdb Titan products into format which can be rendered by CIDD

Ltg2Symprod	_Ltg2Symprod.THOR	Converts Spdb lightning products into format which can be rendered by CIDD
Metar2Symprod	_Metar2Symprod.surface	Converts Spdb surface data products into format which can be rendered by CIDD
Janitor	_Janitor+LogDir _Janitor+FastDelete _Janitor+KeepOut	Recurses through data directory tree and compresses and deletes data.
DsFileDist	_DsFileDist+ncar2Mit _DsFileDist+MitDisplay2ncarRelay	Data distribution
DsSpdbServer	_DsSpdbServer.distrib+ncar2Mit _DsSpdbServer.search+soundings	Spdb data distribution Spdb data server
DsMdvServer	_DsMdvServer.static _DsMdvServer.res1K	Mdv data server

Once you have defined the necessary data directories and any associated parameter files, you are ready to create the data directories and install the run-time parameter files.

Login as **root** on each host and perform the following steps:

1. **mkdir /d1/fieldData**
  2. **chown nowcast /d1/fieldData**
  3. **chgrp shared /d1/fieldData**
- As user **nowcast**:
1. **cd \$RAP\_DIR**
  2. **cvs co make\_include**
  3. **cvs co make\_bin**
  4. **cd \$RAP\_SHARED\_DIR**
  5. **cvs co libs/perl5**
  6. **cd /nfs/ncar/rapShared/libs/perl5/src**
  7. **make install\_shared**
  8. **niwot\_mkdata** to create data dirs on every host or **host\_mkdata** if you want to do each host separately.

Check each host to see that the data directories were made properly and that the param files for the data directories were installed.

## **Building Libraries & Applications**

### Create & Check in Project-Specific Make Files

1. Login as user **nowcast** on any project machine
2. **cd \$RAP\_SHARED\_DIR**
3. **cvs co libs/Makefile**
4. **cvs co apps/Makefile**
5. **cd /nfs/ncar/rapShared/libs**
6. **cp Makefile Makefile.<\$PROJECT>**
7. **Change** the contents of the project-specific make file **Makefile.<\$PROJECT>** to include those libraries which you want to build for the field project.
8. **cd /nfs/ncar/rapShared/apps**
9. **cp Makefile Makefile.<\$PROJECT>**
10. **Change** the contents of the project specific make file **Makefile.<\$PROJECT>** to include those application trees which you want to build for the field project.
11. Check all of your project specific make files into cvs.

### Checkout Source Code

1. **cd \$UTILS\_HOME/bin**
2. **Change** the contents of the file **niwot\_checkout** to reflect the libraries and applications that were just added to the project-specific make files.
3. **niwot\_checkout**
4. Create project-specific make files within each of the application **src** subdirectories. Setting up these application-level make files reduces the number of libraries that you will need to check out and compile for the system and reduces the number of build errors that you are likely to encounter.
5. Check all of your project specific make files into cvs.

Now you are ready to build all of the software used by the Auto-Nowcast system. Because of library dependencies, building the necessary software is often an iterative process. Once you start building the code, you might identify missing libraries or apps and have to edit the **Makefile.<\$PROJECT>** file for the libraries, apps or modify the **niwot\_checkout** script to include the software you need.

There are several scripts in the directory **\$UTILS\_HOME/bin** for building the software in various combinations -- libraries only, applications only, with or without a cvs checkout, etc. Each of these scripts relies on the underlying build script **\$NIWOT\_DIR/niwot\_build** with varying command line options.

If you want to use the **niwot\_build** script to check out, build, and install your libraries and applications AND you have already created your project specific makefiles, try **niwot\_build -m -bld -d**. This will

1. install project specific makefiles
2. build and install the source code
3. print debug messages

If you want to do these steps by yourself and check the results of each step before moving on, you can do the following:

### Install Project-Specific Make Files

1. **cd \$RAP\_SHARED\_DIR**

## 2. `installMakefiles --project <$PROJECT>` Build the Software

Install includes:

1. `cd $RAP_SHARED_DIR/incs`
2. `make install_shared`  
Build and install libraries:
  1. `cd $RAP_SHARED_DIR/libs`
  2. `make install_shared_include`
  3. `make opt install_shared`Build and install applications:
  1. `cd $RAP_SHARED_DIR/apps`
  2. `cd tdrp; make install_shared`
  3. `cd ../`
  4. `make opt install_shared`

### Modifying Parameter Files

---

The next step in configuring the Auto-Nowcast environment is a fairly tedious one. It requires a large number of parameter files and scripts to start the applications to be modified. **This is a very critical step.** Although the applications may seem to run successfully, errors which are very difficult to diagnose later on will indeed occur if the parameter files for each application are not modified correctly.

#### Radar Information files and `radar_env.lookup`

The `radar_env.lookup` is a file whose entries contain radar id/ host (of radar data) pairs and a key word used to identify the pair. The file has three different uses:

1. It is used by the `host_startup` script to generate host process lists. The lookup key word can be used in the pre-processed process lists in `$CONTROL_DIR/proc_list` with the syntax of preceding and following the key word by '?' symbol. At runtime, the key word will be replaced by the Radar Id. in the runtime process list which is placed in `$HOME/controlLocal/runtime`.
2. It is used by application start scripts find radar id, or radar Id/radar host pairs. This enables parameter files to be generic.
3. It is used by `SysViewExpand` to substitute radar id and radar data hosts into generic Sysview diagrams.

Modify relevant param files and start scripts:

1. `cd $CONTROL_DIR/params`.
2. Modify `$CONTROL_DIR/params/radar_env.lookup`. Enter all radar id/radar data host pairs and a key word for each pair.
3. `cd $RADAR_HOME/bin`.
4. Modify all start scripts that use `radar_env.lookup`. Generally these will have the suffix \*.  
`radarLookup`. Make sure that the proper keys are being set and used in the `niwot perl library` lookup commands.
5. `cd $CIDD_HOME/bin`.
6. Similarly modify all start scripts that use `radar_env.lookup`.  
Radar information param files are located in `$RADAR_HOME/params`. They set environment variables specific to the radar including radar id, latitude, longitude, altitude and other variables. These files are sourced by scripts which start applications that require radar specific information. This enables us to keep the parameter files for such applications generic and reduce the number of parameter files that must be maintained for a project using more than one radar data stream.

1. **cd \$RADAR\_HOME/params**
2. For each radar being used in your project, create a radar.<radar\_id> param file. **Note:** The suffix of a radar info filename should match the <radar\_id> of a corresponding entry in **radar\_env.lookup**. The radar start scripts make the assumption that they will be the same. Also, There should be a one to one correspondence between each entry in **radar\_env.lookup** and the radar information files created in **\$RADAR\_HOME/params**.

#### Configure Application Parameters

The application parameters which need to be modified vary depending on which applications are being used for a particular project. Perhaps the best way to approach this task is to methodically go through each of the parameter files for each application in the autoNowcaster one by one, making sure to take a look at each parameter to determine its role. In many cases a parameter will need to be tuned by the scientists; in other cases a configuration change (like change in domain size, grid resolution, adding another algorithm or deleting one) will necessitate a change in the parameters. **Note:**In general the applications in the autoNowcaster contain a **-print\_params** option. The command you would use to print a param file with default parameter settings is **<application name> -print\_params >& <param file name>**

#### Build Project-Specific Map Database

Project maps are used by three different displays: CIDD, TITAN, and RDI. RAP's map database resides in **http://www.rap.ucar.edu/maps**. Any maps required by the Auto-Nowcast field project should be copied from the RAP database and installed in **\$PROJ\_DIR/maps**. Other maps specific to the new field project will need to be fetched from outside sources or digitized.

#### Build Project-Specific CIDD Colorscale Database

The colorscale database for **CIDD** is **\$CIDD\_HOME/colorscapes**. RAP's colorscale database resides in **http://www.rap.ucar.edu/colorscapes**. Any colorscales required for **CIDD** should be copied from the RAP database and installed in **\$CIDD\_HOME/colorscapes**. Colorscales specific to the new field project might have to be created if not available in RAP's colorscale database.

#### Configure Data Distribution

Three applications which are useful for data distribution are **DsSpdbServer** for SPDB data, and the **DsFileDist / DsFCopyServer** pair for MDV data. The parameter files for these servers should ultimately reside in the directory containing the data to which they pertain. In configuring the distribution, create the files in **\$DATA\_HOME/params**, add them to the proper data list in **\$DATA\_HOME/data\_lists** and they will be installed in the proper data directories with **host\_mkdata** or **niwot\_mkdata** commands

#### Configure Data Compression and Scrubbing

The **Janitor** is an application which continually recurses through the data tree and is typically used for compressing data files to save disk space and scrubbing those files determined to be old( a configurable parameter for each data set) and not in a time window specified for saving data by the **\$DATA\_HOME/params/events\_list**. The run-time behavior of the **Janitor** is determined by the parameter files that reside in the data tree. Typically the top level **\_Janitor** param file is very complete and specifies the entire set of **Janitor** application parameters. Lower level **\_Janitor** files should contain only those few parameters which differ from and override the upper level parameter settings. In configuring **\_Janitor** param files, create the files in **\$DATA\_HOME/params**, add them to the proper data list in **\$DATA\_HOME/data\_lists** and they will be installed in the proper data directories with **host\_mkdata** or **niwot\_mkdata** commands.

#### Configure Data Servers To Run On Each Host

The four data service applications which typically run on all machines in the Auto-Nowcast system are:

1. **DsServerMgr**
2. **DataMapper**
3. **Scout**
4. **Janitor**

These applications typically appear in the process list

file **\$CONTROL\_DIR/proc\_list/EVERY\_HOST.always** so you shouldn't have to make any modifications to enable these. You may, however, need to modify this approach if your system configuration includes an exposed data relay host for data distribution where running **DsServerMgr** could be considered a security risk.

If you are distributing data to/from an exposed host outside the firewall, you may want to run the server manager in secure mode using: **DsServerMgr -secure** or avoid running **DsServerMgr** altogether to minimize the risk of automatically starting up unexpected server processes. Instead of running the server manager you could use the exposed host strictly as a data relay machine and limit the use of that machine to the following processes maintained by **auto\_restart** :

1. **DsFileDist**
2. **DsFCopyServer**
3. **DsSpdbServer**
4. **Janitor**
5. **auto\_restart**

If you take this approach of explicitly running the Ds-applications via the auto restarter, then you might need to modify the process list for **EVERY\_HOST.always** and take out **DsServerMgr**, **DataMapper**, **Scout**, and **Janitor** and add them to the process lists on a host by host basis.

#### Setup Nightly Status Email Lists

1. **cd \$CONTROL\_DIR/params**
2. **Change** the email addresses in the file **email.list**. These email addresses will receive two nightly emails:
  1. 24-hour summary of the process control status
  2. machine status on uptime, disk space, and memory usage
3. **cd \$CONTROL\_DIR/crontab**. Edit the **CONTROL\_HOST** crontab file and add the following:

```
#  
# Run the reliability statistics once a day. This should be  
# run on the CONTROL HOST (atNcar), just after midnight.  
#  
01 00 * * * csh -c "niwot_status >& /dev/null"  
#  
# Run the host machine status check once a day  
#  
59 23 * * * csh -c "query_hardware_status >&/dev/null"
```

---

### **Setting up a System Monitor**

At this point everything should be in place for developing a complete **SysView** diagram of the system configuration. If the **SysView** diagram has not already been developed for planning purposes, it is a very good idea to put one together for system documentation and realtime monitoring.

As user **nowcast**:



1. `cd $SYSVIEW_HOME/bin/sysview2`
2. `java -jar sysview2.jar`
3. Create diagrams for each **\$HOST** and save these diagrams in **\$SYSVIEW\_HOME/diagrams**.

Note: You can use nowcast environment variables in your application hosts and data paths. You must write your environment variables with added parenthesis as displayed in the following example: **\$INGEST\_HOST** must be written **\$(INGEST\_HOST)**. You can also substitute lookup key words for radar ids from the **\$CONTROL\_DIR/params/radar\_env.lookup**. You must write these variables with added **?** as displayed in the following example: key word **radar1** must be written **?radar1?**. All these variables will be resolved by a program called **SysViewExpand** which will be executed when you run **SysView** with the command `run_Sysview`.

Also note that SysView relies on the **DataMapper** application to get information in realtime about the datasets on various hosts. Therefore you must be running the **DataMapper** on hosts for which you want information about the local data.

### **Ready, Set, Go...**

---

By now the Auto-Nowcast system should be ready to begin testing the startup and shutdown procedures. This step will let you know if you have properly set up the start scripts, parameter files, process lists, data directories, etc.

A prudent approach to testing components of the system would be to manually start each application as you complete configuration of its parameter file. If you are feeling more daring, you can start up processes along with the **auto-restartservices**.

To start up a single machine in the system:

1. login as user **nowcast** on the machine of interest
2. **host\_startup realtime**

If you want to be brave and kick off the whole system at once:

1. login as user **nowcast** on the **\$CONTROL\_HOST**
2. **niwot\_startup realtime**

When starting the whole system via **niwot\_startup**, you should see each of the project hosts reporting back to the console window of the **\$CONTROL\_HOST** as they startup.

The corresponding commands for shutting down the Auto-Nowcast system are:

### **host\_shutdown and niwot\_shutdown**

Some tools you can use to determine if your hosts started up properly are:

1. **print\_procmmap -up -hb -c 5**  
This will tell you all of the processes which are registering with the process mapper called procmmap . Make sure that all of the processes which should be running are showing up in this list.
2. Use **tail -f \$CONTROL\_LOCAL\_DIR/runtime/auto\_restart.log**  
to see which process keep restarting on the host. This indicates that there is a problem with the start

script, the param file for the application, perhaps the instance name in the process list doesn't match the one in the start script, or the application executable is missing.

3. Execute **run\_Sysview** (assuming your **SysView** diagrams are set up properly).

### **Freezing the Source Code**

---

At some point once you are satisfied with the basic workings of the Auto-Nowcast applications, you will have to do a code freeze. This amounts to doing one final CVS checkout and build of all the software as described in [Building Libraries & Applications](#). There's probably a lot that could be said here about both the importance and the potential pitfalls of a code freeze. Suffice it to say:

1. Don't wait until the day before you start [Packaging up the Field Project](#) before you do a code freeze.
2. Be circumspect when accepting any new code after the code freeze.

**IMPORTANT:** Once you are certain that no further updates or changes will be incorporated into the source code, mirror the libraries and applications (source and binaries) from the machine which is hosting the [Shared Project Space](#) to the **\$FIELD\_SUPPORT\_HOST**. Maintaining a copy of the frozen source code which is operating in the field will allow you to debug and possibly correct software bugs which arise during operations without doing a full software upgrade during the field season.

### **Packaging up the Field Project**

---

Switch to the Field Host Names and IP Addresses

If you have been doing a setup at RAP on the actual field machines that are to be shipped out into the field, only the domain name and IP addresses of the machines will change. The hostnames should not change. In this case you will only need to update those project files in which domain names were specified:

1. **Change** the system domain name and IP address.
2. **Reconfigure** the second ethernet card of the **\$INGEST\_HOST** for the new IP address.
3. **Change** the file **\$PROJ\_DIR/.shosts** to the new domain name.

If instead of setting up the Auto-Nowcast system on the actual field machines, you have been doing a "mock-up" using various machines around RAP, all of the explicit hostnames in the project files will have to be changed in addition to the domain name and IP address changes made above.

In most cases, logical hostnames are used via UNIX environment variables. However, in some project files explicit hostnames must be specified. Refer to [Modifying Parameter Files](#) for the most complete explanation of the project files which should be examined. A quick list of the files in which explicit hostnames might be used include:

1. **\$HOME/.fvwsrc.\***
2. **\$CONTROL\_DIR/params/host\_env.\***

Do Final CVS Checkin of the Project

1. **cd \$PROJ\_DIR**
2. **cvs -n update**
3. **cvs commit**

Burn a CD-ROM of the Project

On at least one occasion, a PC has been shipped from RAP and has gone missing in the mail. This is particularly awkward when the machine in question has all the frozen source code and/or executable files

on it. It is a good idea to write all the executables, the binary libraries, the applications source tree and the library source tree to a CD-ROM for backup purposes.

#### Make Final System Changes

1. **BE SURE** to remove any local cross mounted file systems!
- 2.
3. **Switch** from DNS lookup to host table lookup. The system's administrators don't necessarily like to do this, but have 'em do it anyway!!!

#### At the Field Site

---

1. Make sure **ssh** authentication still works with the new domain name. See [Modifying & Installing .dot Files](#) for a brief explanation of **ssh** authentication.
2. Make sure the cross mounts are set up properly for the [Shared Project Space](#).
3. Make sure that the NTP (network time protocol) service is working properly for all machines.
4. Change the **nowcast** user password to a project-specific external password. Do **NOT** use the internal password for user nowcast on the project machines out in the field.
5. On the LDM servers, change the pqact.conf to point to the field project radars!
6. If you are maintaining previous season data, you may need to merge the previous data into the current data directory structure.

# Monitoring

## Daily Checklist

## Weekly Checklist

### Daily Checklist

---

On a daily basis, the operator should perform the following tasks:

1. Check disk and memory usage
2. Check process status from the previous day
3. Check current process status
4. Check current data status
5. Monitor the system
6. Check email from cron jobs
7. Check data archiving mechanism

#### Check disk and memory usage

The various Auto-Nowcast Hosts must have sufficient disk and memory available to run the Auto-Nowcast Environment processes and store data files.

On a daily basis, a utility called **niwot\_status** is run on **\$CONTROL\_HOST** to check the disk usage, uptime, and memory statistics on all the Auto-Nowcast Hosts. The results are emailed to addresses in **\$CONTROL\_DIR/params/email.list**. The status script is run nightly as a crontab job. If you want to receive daily mail regarding the status of the Auto-Nowcast Hosts, add your email address to the email list file.

For any Auto-Nowcast Host you can check the disk usage and memory usage at any time. The UNIX command **df** is used for checking disk usage. To check memory usage, either run **top** or **cat /proc/meminfo**.

#### Check process status from the previous day

On a daily basis, a utility called **query\_hardware\_status** is run to check process status for the day. The results are emailed to addresses in **\$CONTROL\_DIR/params/stats\_email.list**. The status script is run nightly as a crontab job. If you want to receive daily mail regarding the daily process status, add your email address to the email list file.

The process status files are also date/time stamped and stored on the **\$CONTROL\_HOST**. To check the process status from a particular day, examine the corresponding file in **\$RAP\_DATA\_DIR/\$PROJECT/other/reliability\_stats/YYYYMMDD\_procmmap.stats**. Keep in mind that the process status files get scrubbed by the Janitor along with other files and datasets in the Auto-Nowcast Environment.

#### Check current process status

**SysView** monitors the status of the Auto-Nowcast Environment datasets and processes. The User's Manual for SysView can be downloaded in \*.pdf format. The procmmap is also a useful utility to monitor the status of a given host in the Auto-Nowcast Environment.

If you suspect problems with any of the processes, see the Process Control FAQ for suggestions on diagnosing and correcting the problems.

### **Check current data status**

**SysView** monitors the status of the Auto-Nowcast Environment datasets and processes. SysView can be downloaded in \*.pdf format.

If you suspect problems with any of the datasets, see the Data Ingest & Management FAQ for suggestions on diagnosing and correcting the problems.

### **Check email from cron jobs**

To check the results from any cron jobs that are running at your installation, read the email for the nowcast user on the **\$CONTROL\_HOST**. Most cron jobs are run near midnight, so checking email every morning is a good practice.

### **Check data archiving mechanism**

## **Weekly Checklist**

---

On a weekly basis, the operator should perform the following tasks:

1. Update the events list
2. Backup data to tape

### **Update the events list**

Update the meteorologically interesting case dates in the file **\$DATA\_HOME/params/events.list**. The cases in this file will not be scrubbed off the disk by the Janitor off the disk.

### **Backup data to tape or utilize other archiving mechanism**

# Data Architecture

## RAL Internal Data Formats RAL Data Directory Structure

### Data Sources

---

The availability of various data sources at a particular Auto-Nowcast Environment installation will dictate the number and types of algorithms that can be run. The minimum data required for the Auto-Nowcast Environment is NEXRAD radar data. A typical configuration for the Auto-Nowcast Environment might include the following input data streams:

- NEXRAD radar data
- Satellite data
- Soundings
- Surface observations

Although the mechanism for acquiring the various external data sources may vary from one Auto-Nowcast Environment installation to another, the data must conform to data formats known to the applications in the Auto-Nowcast Environment.

### RAL Internal Data Formats

---

1. **MDV** (or **Meteorological Data Volume**): This is 3D or 2D gridded data. For example radar data volumes are stored in **MDV** format. Terrain data is stored in **Mdv** format.
2. **SPDB** (or **Symbolic Product Data Base**): This is a format for point data. For example, Lightning data and surface observations are stored in **SPDB** format. Titan storms can be converted to **SPDB** format.
3. **FMQ** (or **File Message Queue**): This is not so much of a data type but a fixed size file which which applications can use to exchange data of any type or messages. Some of our applications read and write data to and from these circular queues and therefore they are managed with our data.
4. **titan storm track**: This is the output format of storms generated by TITAN.
5. **DsRadar**: This is RAL's format for radar beam data.

### RAL Data Directory Structure

---

The head of data hierarchy at RAP is **\$RAP\_DATA\_DIR**. This is an environment variable which is set in the project **.cshrc**. The data servers which read and write data for the applications will assume all data paths that do not begin with a **'/'** are relative to **\$RAP\_DATA\_DIR** and all data paths that begin with a **'/'** are absolute paths.

Under **\$RAP\_DATA\_DIR**, data is divided by type. For example, typical subdirs of **\$RAP\_DATA\_DIR** might include:

- **fmq**
- **mdv**
- **raw**
- **spdb**
- **titan**

**fmq** directory and file structure:

The **fmq** directory contains the File Message Queues which have suffix **.buf** and corresponding **.stat** file. Typical **FMQ** files have the following paths relative to **\$RAP\_DATA\_DIR**:

**fmq/<fmq name>&GT;.buf**

**fmq/<fmq name>&GT;.stat**

**mdv** directory and file structure:

**mdv** subdirectories are further divided by dataset name, and date. The dataset filename is the valid time of the data in hours, minutes and seconds. Following is the structure of an **MDV** dataset path relative to **\$RAP\_DATA\_DIR**:

**mdv/<data set name>/yyyymmdd/hhmmss.mdv**

**raw** directory and file structure:

**raw** or sometimes called **other** subdirectories are further divided by dataset name but the rest of the directory structure depends on the raw data and its format.

**spdb** directory and file structure:

**spdb** subdirectories are further divided by dataset name and these directories contain the database files. The files are separated by day with basename **yyyymmdd** and for each day there is a **\*.data** file and a **\*.index** file. The database files for one day have the following paths relative to **\$RAP\_DATA\_DIR**:

**spdb/<data set name>/yyyymmdd.data** and

**spdb/<data set name>/yyyymmdd.index.**

**titan** directory and file structure:

**titan** subdirectories are further divided by dataset name and these directories contain the database files. The files are separated by day with basename **yyyymmdd** and for each day there is a **\*.sd5** file, a **\*.sh5** file, **\*.td5** file, and a **\*.th5** file. The database files for one day have the following paths relative to **\$RAP\_DATA\_DIR**:

**titan/<data set name>/yyyymmdd.sd5**

**titan/<data set name>/yyyymmdd.sh5**

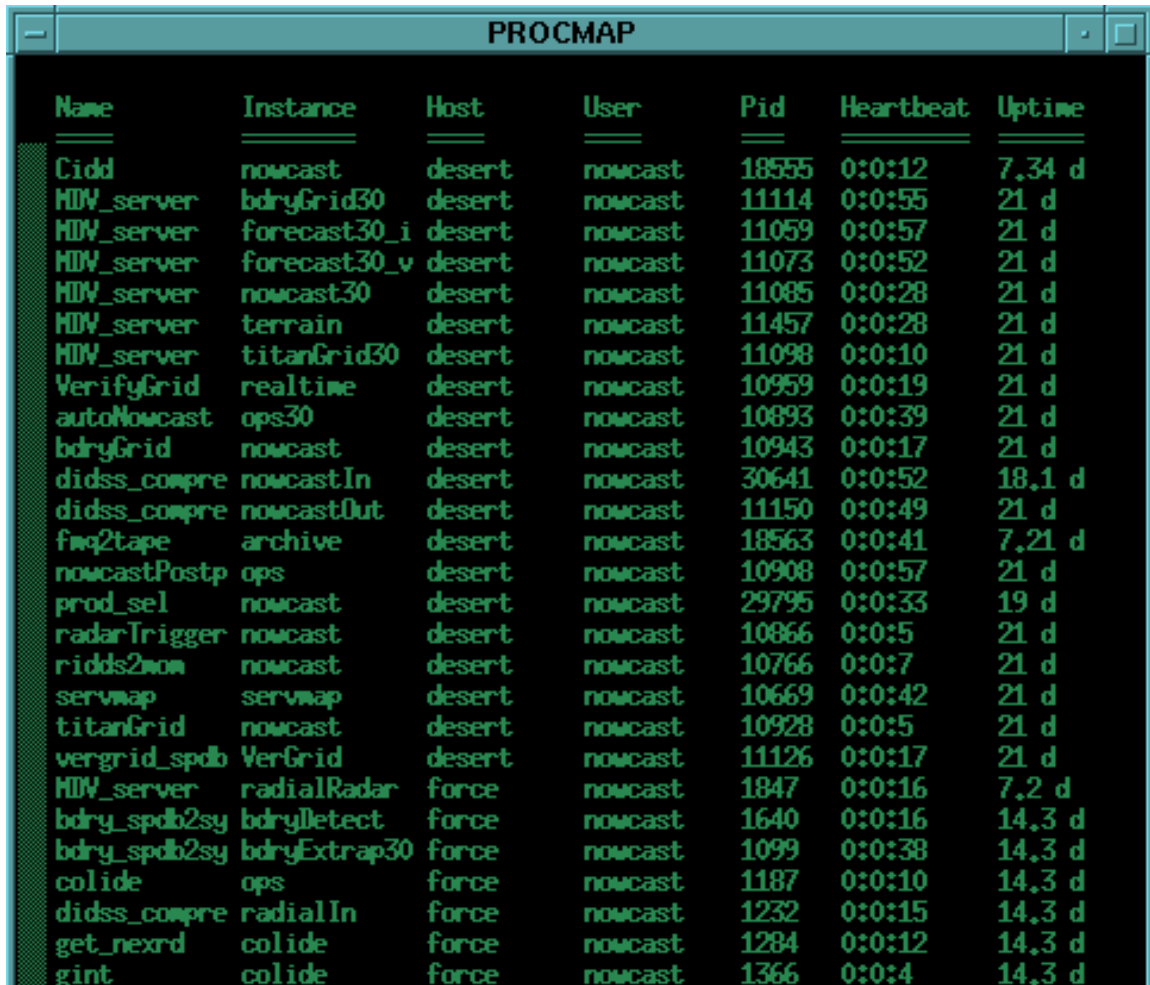
**titan/<data set name>/yyyymmdd.td5**

**titan/<data set name>/yyyymmdd.th5**

All data files in the Auto-Nowcast Environment are timestamped based on UTC, a.k.a, GMT time.

## Procmap and the Auto-restarter

The process mapper, a.k.a. *PROCMP*, is a program which stores state information on the processes which are currently running in the Auto-Nowcast Environment. The procmap window (shown below) displays the current process status.



The screenshot shows a window titled "PROCMP" with a table of process status. The table has seven columns: Name, Instance, Host, User, Pid, Heartbeat, and Uptime. The data is as follows:

Name	Instance	Host	User	Pid	Heartbeat	Uptime
Cidd	nowcast	desert	nowcast	18555	0:0:12	7,34 d
MDV_server	bdryGrid30	desert	nowcast	11114	0:0:55	21 d
MDV_server	forecast30_i	desert	nowcast	11059	0:0:57	21 d
MDV_server	forecast30_v	desert	nowcast	11073	0:0:52	21 d
MDV_server	nowcast30	desert	nowcast	11085	0:0:28	21 d
MDV_server	terrain	desert	nowcast	11457	0:0:28	21 d
MDV_server	titanGrid30	desert	nowcast	11098	0:0:10	21 d
VerifyGrid	realtime	desert	nowcast	10959	0:0:19	21 d
autoNowcast	ops30	desert	nowcast	10893	0:0:39	21 d
bdryGrid	nowcast	desert	nowcast	10943	0:0:17	21 d
didss_compre	nowcastIn	desert	nowcast	30641	0:0:52	18,1 d
didss_compre	nowcastOut	desert	nowcast	11150	0:0:49	21 d
fwq2tape	archive	desert	nowcast	18563	0:0:41	7,21 d
nowcastPostp	ops	desert	nowcast	10908	0:0:57	21 d
prod_sel	nowcast	desert	nowcast	29795	0:0:33	19 d
radarTrigger	nowcast	desert	nowcast	10866	0:0:5	21 d
ridds2nom	nowcast	desert	nowcast	10766	0:0:7	21 d
servmap	servmap	desert	nowcast	10669	0:0:42	21 d
titanGrid	nowcast	desert	nowcast	10928	0:0:5	21 d
vergrid_spdb	VerGrid	desert	nowcast	11126	0:0:17	21 d
MDV_server	radialRadar	force	nowcast	1847	0:0:16	7,2 d
bdry_spdb2sy	bdryDetect	force	nowcast	1640	0:0:16	14,3 d
bdry_spdb2sy	bdryExtrap30	force	nowcast	1099	0:0:38	14,3 d
colide	ops	force	nowcast	1187	0:0:10	14,3 d
didss_compre	radialIn	force	nowcast	1232	0:0:15	14,3 d
get_nexrd	colide	force	nowcast	1284	0:0:12	14,3 d
gint	colide	force	nowcast	1366	0:0:4	14,3 d

Each line in the procmap window identifies a separate process in the Auto-Nowcast Environment.

- In the *Name* column is a list of the executables (or applications) which make up the Auto-Nowcast Environment.
- *Instance* is an additional descriptor which allows for unique identification of the executable processes. For example, the MDV\_server appears multiple times in the process mapper, but the instance name distinguishes between each MDV\_server by indicating on which dataset the server is operating.
- The *Host* column identifies the machine on which the process is running. It is useful to note that the procmap window orders processes alphabetically by host name first, then by executable name on each host.



- The *User* column indicates the login name under which a process is running. Since the Auto-Nowcast Environment is operated under the "nowcast" account, that username will appear for all of the processes.
- The *Pid* is the UNIX process identifier in the operating system's process table.
- *Heartbeat* indicates the length of time since the process last registered with procmmap. Processes are configured to register once every minute.
- *Uptime* shows the length of time since the process was last restarted.

PROCMP works together with the *auto-restarter* to keep processes up and running in the Auto-Nowcast Environment. The role of the auto-restarter is to make sure that all of the required processes are running and registering a regular heartbeat with the process mapper. Any process which is missing from the procmmap or which has not registered a recent heartbeat is killed and restarted by the auto-restarter.

The auto-restarter makes a full check of the process list every minute. Any process which gets restarted is entered into a log file which is summarized nightly and mailed to the operator for [daily maintenance checks](#).