Contents

21 Jan 2015

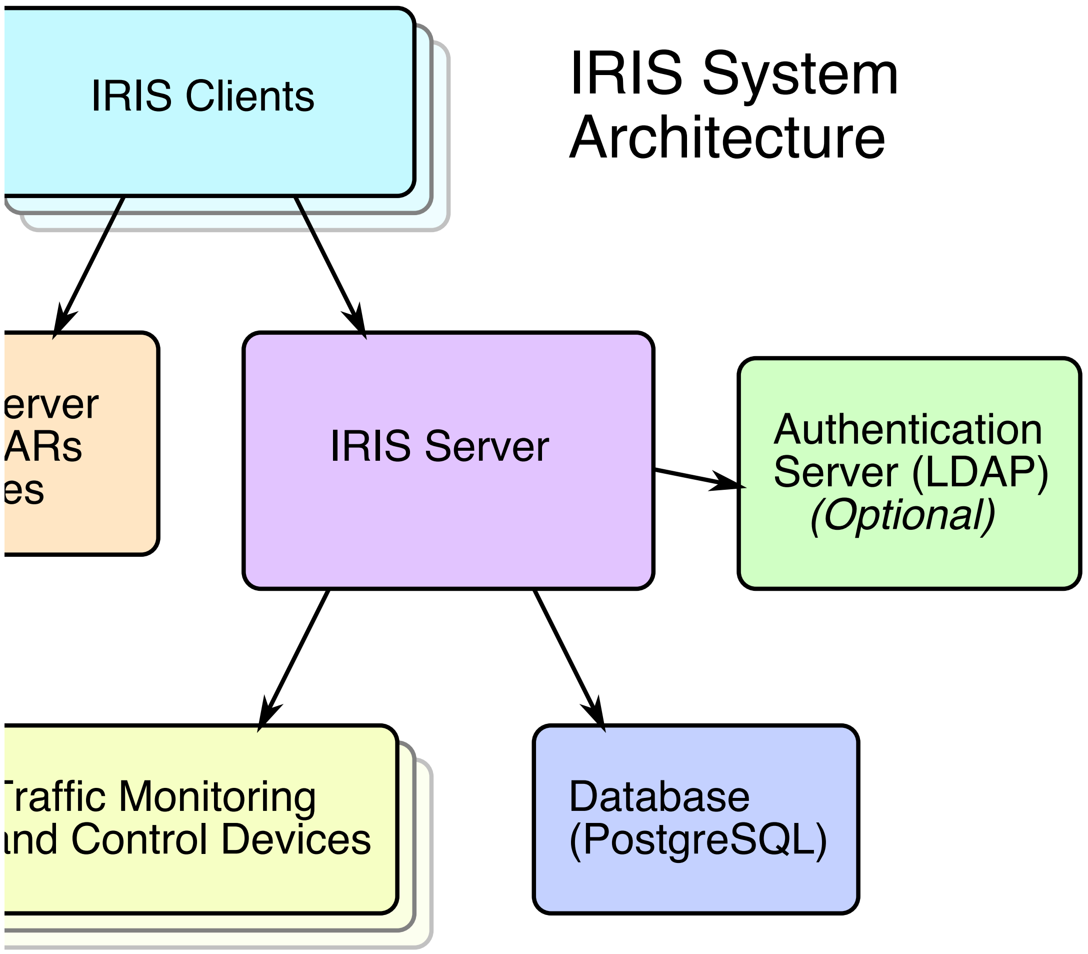
IRIS Administrator Guide

1 Overview

IRIS is an open source advanced traffic management system. It provides an integrated platform for transportation agencies to manage traffic monitoring and control devices. The software is written in Java and licensed for anyone to use under the GPL. In addition, all dependencies required to install and operate an IRIS system are available as free software. IRIS stands for Intelligent Roadway Information System.

The IRIS software presents an intuitive map-based interface to system operators. This user interface has been refined over many iterations by getting feedback from operators to streamline their workflow.

1.1 System Architecture



The software has a client/server architecture. System configuration data is stored in a PostgreSQL database, and managed by the IRIS server. The server also handles communication with all traffic control and data collection devices. The client software is distributed by an Apache web server, using Java Web Start. All communication between the server and clients is encrypted using secure socket layer (SSL). The server may be configured to pass authentication requests off to an external LDAP server, allowing IRIS to integrate into an existing authentication system.

1.2 Device Types

IRIS is able to manage several different types of traffic monitoring and control devices. These include:

- DMS (dynamic message signs)
- CCTV cameras and Video switchers
- VDS (vehicle detection systems)
- LCS (lane-use control signs)
- Ramp meters
- Gate Arms
- Beacons
- RWIS (road-weather information systems)
- In-road lighting (dynamic striping)

1.3 Protocols

Many different protocols are supported for communicating with the various device types. Additional protocols can be added for any existing device type by programming a simple driver interface. These are some existing protocols:

- NTCIP class A, B and C (DMS)
- DMS-XML (DMS)
- MnDOT 4- and 5-bit (VDS, ramp meters, LCS, beacons, IRL)
- Wavetronix SS105 and SS125 (VDS)
- Canoga Traffic Sensing System (VDS)
- RTMS G4 (VDS)
- Pelco D (pan/tilt/zoom)
- Infinova (pan/tilt/zoom)
- Cohu (pan/tilt/zoom)
- AD "manchester" (pan/tilt/zoom)
- Vicon (pan/tilt/zoom)
- Msg Feed (external system DMS messages)
- Pelco Video Switcher
- HySecurity STC (Gate Arms)
- Optical Scientific ORG-815 (precipitation)
- SSI RWIS sensor
- DLI DIN Relay (LCS)

1.4 Advanced Features

IRIS also contains some higher-level traffic management features:

- Continuously updated traffic map (density, speed or flow)
- Scheduled deployment of control devices
- Incident management

- Travel time estimation for traveler information
- Variable speed advisories
- Coordinated ramp metering strategies

2 Installation

2.1 Operating System

Install Fedora onto the server computer. Other operating systems, such as Red Hat Enterprise Linux, can also be used, but the procedures may vary slightly. This machine should have a minimum of 2 GB of RAM and 200 GB hard drive storage space. The IRIS server does not have a GUI interface, so this computer can be set up headless.

After the operating system is installed, make sure the hostname is configured correctly in DNS. The hostname is needed for some configuration files in the following steps. The following command should display the hostname that can be used to access the server:

```
hostname -f
```

2.2 Install IRIS

Download and install the IRIS package. This should be a file called **iris-*{major}.{minor}.{micro}-{build}.{arch}*.rpm**. The *{major}*, *{minor}*, and *{micro}* version numbers and the *{build}* number can change with each release. The system architecture, *{arch}*, will be *noarch*, since Java is architecture independent. It can be installed with the following command (as root):

```
yum install iris-{major}.{minor}.{micro}-{build}.r
```

Yum will install dependencies not already installed, including OpenJDK, PostgreSQL and Apache.

2.3 Initialize IRIS

Next, you must initialize the IRIS server installation. Run the following command (as root):

```
iris_ctl init
```

This command will perform the following steps:

1. Update the `/etc/iris/iris-client.properties` file with the server's hostname.
2. Create an SSL key pair for encrypted communication between the server and clients. The keystore files are `/etc/iris/iris-server.keystore` and `/etc/iris/iris-client.keystore`. The keystore passwords are put into `/etc/iris/iris-server.properties` and `/etc/iris/iris-client.properties`.
3. Create a symbolic link to the PostgreSQL JDBC driver to make it available to the IRIS server.

4. Create the database cluster and start the PostgreSQL server.
5. Create the "tms" PostgreSQL user, which IRIS uses to connect to the database.
6. Create the "tms" database and populate it using a template SQL script.
7. Configure the apache (httpd) and IRIS services to start automatically.
8. Create symbolic links to the current IRIS software version.

The command should finish with the following message:

```
Successfully initialized the IRIS server
```

2.4 Server Properties

The IRIS server has a configuration file which controls some of its basic settings — */etc/iris/iris-server.properties*. Most of the default settings should work, but it may be necessary to make some changes. The file can be modified using a text editor, such as *gedit* or *vi*.

2.4.1 Internationalization

There are three properties which control internationalization (i18n). They are *language*, *country* and *variant*. These should only be changed if the IRIS software has been localized for a specific locale.

2.4.2 District

This property is used when multiple IRIS servers are running within the same organization.

2.4.3 HTTP Proxy

These properties can be used when IRIS must connect to HTTP servers using a proxy server, due to organization firewall policies.

2.4.4 Database Connection

The *db.url*, *db.user* and *db.password* properties control how the IRIS server connects to the PostgreSQL database. None of these properties should be changed, since they were configured earlier by the `iris_ctl` script.

2.4.5 SONAR Configuration

The *sonar.ldap.urls* property can be used to let IRIS pass user authentication requests to one or more LDAP servers. Multiple URLs must be separated by a space. IRIS can manage user accounts and passwords without an LDAP server, but this feature allows operators to log into IRIS without requiring them to remember an additional user name and password. If your organization already has an LDAP server, such as Active Directory, you should use that.

The URL protocol is normally *ldap:*, but must be *ldaps:* for encrypted communication over SSL. To use ldaps, you must first import the SSL certificate into the IRIS keystore. Once you have obtained the SSL certificate for the LDAP server, save it as *ldap.cert*. Now you can import it with the following command (as root):

```
keytool -import -alias ldap-cert -keystore /etc/ir
```

The *keystore.password* is used for accessing keys in `/etc/iris/iris-server.keystore`. This password is automatically generated by the `iris_ctl` script. It will not need to be changed.

2.5 Start Services

Finally, the IRIS server can now be started with the following command:

```
systemctl start iris.service
```

Check that everything started OK:

```
tail -f -n 40 /var/log/iris/iris.stderr
```

2.6 Initial Login

From a client computer with Java installed, point your web browser at http://YOUR_SERVER_NAME/iris-client/. From there, you should be able to launch the IRIS client Web Start application and log in.

Enter `admin` as the username and `atms_242` as the password for the initial login. After creating and logging in with a real administrator account, the `admin` user account should be disabled.

3 Basic Setup

3.1 System Attributes

A **system attribute** is a configurable value which can be used to customize an IRIS system. Select the `View ➔ System ➔ System Attributes` menu item to display the System Attributes form. Each system attribute has a corresponding type: string, boolean, integer or floating point number. String values are limited to 64 characters or less. Boolean values must be specified as either "true" or "false". Many numeric system attributes are constrained to a range of values within a minimum and maximum bound, depending on the attribute. A system attribute can be customized by updating the value in the table on the System Attributes form. When a system attribute has been modified from its default value, it will be displayed **in bold**. Most attribute changes take effect immediately, but some require the IRIS client or server to be restarted. Some System Attributes are described in more detail below.

3.1.1 map_segment_max_meters

This is used for two distinct purposes: 1) specifying the maximum distance between adjacent stations to draw segments on the map and 2) specifying the maximum distance from a station to another downstream station for associating density, speed, etc. data with a segment.

3.2 Client Properties

The IRIS client has a configuration file (stored on the server) which controls some of its basic settings — */etc/iris/iris-client.properties*. It can also contain default values for some user properties that will be used in the event that they are unspecified in the user properties file.

language
: ISO 639 alpha-2 or alpha-3 language code. E.g., "en", "es", "fr".

country
: ISO 3166 alpha-2 country code or UN M.49 numeric-3 area code. E.g., "US", "MX", "CA".

variant
: IETF BCP 47 language variant subtag.

district
: District name. Useful in cases where multiple IRIS servers exist within the same organization.

proxy.host
: If a proxy is to be used, its hostname can be specified here.

proxy.port
: If a proxy is to be used, its port number can be specified here.

no.proxy.hosts
: A comma-separated list of host IPs/subnets to exclude from proxy use.

keystore.file
: URL for the client keystore file.

keystore.password
: Password for the client keystore.

sonar.host
: IP or hostname of the SONAR server.

sonar.port
: TCP port number of the SONAR server.

tdxml.detector.url
: URL for XML detector stream.

map.tile.url
: Base URL for map tileset. Must end in "/".

video.host
: IP or hostname of video server/proxy.

video.port
: TCP port number of video server/proxy.

autologin.username
: A username to be used for automatic login upon client startup. As this is a global client property, it is generally not useful in production environments.

autologin.password
: A password to be used for automatic login upon client startup. As this is a global client property, it is generally not useful in production environments.

3.3 User Properties

The IRIS client also maintains a file that is used to store client state and user/machine-specific preferences. If this file doesn't exist, the IRIS client will automatically create it. The file is named "user.properties", and can be found in the "iris" folder of the user's home directory. Many of the properties stored in this file are used for the preservation of client UI state (e.g., window geometry, last-selected tab), but others are useful for overriding various defaults:

tab.list

A comma-separated list (whitespace ignored) that specifies the visibility and ordering of the primary UI tabs in the IRIS client. If unspecified, the following default value is used:

```
incident, dms, camera, lcs,
ramp.meter, gate.arm, r_node,
action.plan, comm
```

3.4 Users and Roles

When a user logs into the IRIS system, the user is checked for validity. The user account must be enabled, and be assigned to an enabled **role** for a log in to succeed. If the user has a "distinguished name" (dn), then authentication is performed using LDAP. Otherwise, IRIS checks the password against the stored password hash for that user account. Select the `View ➔ System ➔ Users and Roles` menu item to display the **Users and Roles** form.

The template SQL database contains one administrator role, called `admin`. This role is assigned **capabilities** which allow unfettered access to the IRIS system. Other roles can be created by an administrator, to allow users with a specific set of capabilities, as needed. For example, there are capabilities defined for each of the main tabs on the user interface: camera_tab, dms_tab, lcs_tab, meter_tab, etc. Also, these capabilities can be disabled, preventing any user from having access to them. So, if a system does not contain any LCS devices, the lcs_tab capability could be disabled globally, to prevent that tab from appearing in the user interface for any users. WARNING: changes take effect immediately — if the user_admin capability is disabled or removed from the administrator role, the administrator will lose access to change roles and capabilities.

3.5 Mapping

The IRIS client can use map tiles to display high-quality maps as a background layer. The map tiles are fetched from a web server on demand as the client map is panned and zoomed. The tiles can be generated from data downloaded from OpenStreetMap. The following instructions describe one method of generating the map tiles on Fedora 20.

3.5.1 Get planet file

Download the latest planet file (about 32GB).

```
wget http://planet.openstreetmap.org/planet/planet
```

3.5.2 Download and install PostGIS

```
sudo yum install postgis postgis-docs postgis-util
```

3.5.3 Configure PostGIS and create spatial DB

```
# Create DB named gis
su - postgres
createdb --encoding=UTF8 gis
psql -d gis -f /usr/share/pgsql/contrib/postgis-64

# Edit kernel parameters to increase max size of s
sudo sysctl -w kernel.shmmax=268435456
sudo sysctl -p /etc/sysctl.conf
sysctl -a | grep kernel.shmmax

# Update postgresql config
vim /var/lib/pgsql/data/postgresql.conf
        checkpoint_segments = 20
        maintenance_work_mem = 256MB # 256000 for
        autovacuum = off
        shared_buffers = 4GB
        temp_buffers = 128MB
        maintenance_work_mem = 4GB
service postgresql restart
```

3.5.4 Build and install osm2pgsql from source

```
su - tms
sudo yum -y install git
sudo yum -y install geos-devel proj-devel postgres
sudo yum -y install gcc-c++ protobuf-c-devel autoc
mkdir ~/osm; cd ~/osm
git clone https://github.com/openstreetmap/osm2pgs
cd ~/osm/osm2pgsql
./autogen.sh
./configure
make
sudo make install
```

3.5.5 Import OSM planet file into PostGIS

This process takes a long time (around 30 hours). See performance comparisons here. Determine the bounding box for the region of interest. For Minnesota it is: -97.5, 43.0, -89.0, 49.5.

```
su - postgres
osm2pgsql -v --number-processes=16 -d gis -s --bbc
```

3.5.6 Build and install mapnik library from source

See the mapnik install docs. The Springmeyer docs are also very useful.

```
su - tms
# Install dependencies
sudo yum -y install make gcc47 gcc-c++ bzip2-devel
      zlib-devel libjpeg-devel libxml2-devel pytho
      python-nose python27-devel python27 proj-dev
      proj-nad freetype-devel freetype libicu-deve
```

```
# Install optional deps
sudo yum -y install gdal-devel gdal postgresql-dev
      libcurl-devel libcurl cairo-devel cairo pyca
# Build recent version of Boost
export JOBS=`grep -c ^processor /proc/cpuinfo`
mkdir -p ~/osm/src; cd ~/osm/src
export BOOST_VERSION="1_55_0"
export S3_BASE="http://mapnik.s3.amazonaws.com/dep
curl -O ${S3_BASE}/boost_${BOOST_VERSION}.tar.bz2
tar xf boost_${BOOST_VERSION}.tar.bz2
cd boost_${BOOST_VERSION}
./bootstrap.sh
./b2 -d1 -j${JOBS} \
    --with-thread \
    --with-filesystem \
    --with-python \
    --with-regex -sHAVE_ICU=1  \
    --with-program_options \
    --with-system \
    link=shared \
    release \
    toolset=gcc \
    stage
sudo ./b2 -j${JOBS} \
    --with-thread \
    --with-filesystem \
    --with-python \
    --with-regex -sHAVE_ICU=1 \
    --with-program_options \
    --with-system \
    toolset=gcc \
    link=shared \
    release \
    install
cd ~/osm/src
# Set up support for libraries installed in /usr/l
sudo bash -c "echo '/usr/local/lib' > /etc/ld.so.c
sudo ldconfig
# Build and install mapnik stable branch: 2.3.x
mkdir -p ~/osm/src; cd ~/osm/src
git clone https://github.com/mapnik/mapnik -b 2.3.
cd ~/osm/src/mapnik
./configure
make
make test-local
sudo make install
# Verify everything is good
echo “import mapnik” | python
```

3.5.7 Install mapnik tools

See docs.

```
sudo yum -y install svn
cd ~/osm
svn export http://svn.openstreetmap.org/applicatio
# Install prepared world boundary data
```

```
cd ~/osm/mapnik
mkdir world_boundaries
wget http://tile.openstreetmap.org/world_boundarie
tar xvzf world_boundaries-spherical.tgz
wget http://tile.openstreetmap.org/processed_p.tar
tar xvjf processed_p.tar.bz2 -C world_boundaries
wget http://tile.openstreetmap.org/shoreline_300.t
tar xjf shoreline_300.tar.bz2 -C world_boundaries
wget http://www.naturalearthdata.com/http//www.nat
unzip ne_10m_populated_places.zip -d world_boundar
wget http://www.naturalearthdata.com/http//www.nat
unzip ne_110m_admin_0_boundary_lines_land.zip  -d
```

3.5.8 Patch osm.xml to create iris_osm.xml

```
cd ~/osm/mapnik/
cp osm.xml iris_osm.xml
patch -p0 < ../01_drop_junc_power_layers.patch
patch -p0 < ../02_interstate_shields.patch
patch -p0 < ../03_trunk_shields.patch
patch -p0 < ../04_cty_rd_shields.patch
patch -p0 < ../05_motorway_color.patch
patch -p0 < ../06_trunk_color.patch
patch -p0 < ../07_primary_color.patch
patch -p0 < ../08_secondary_color.patch
patch -p0 < ../09_tertiary_color.patch
```

3.5.9 Copy symbols used for roadway shields

FIXME

3.5.10 Generate tiles

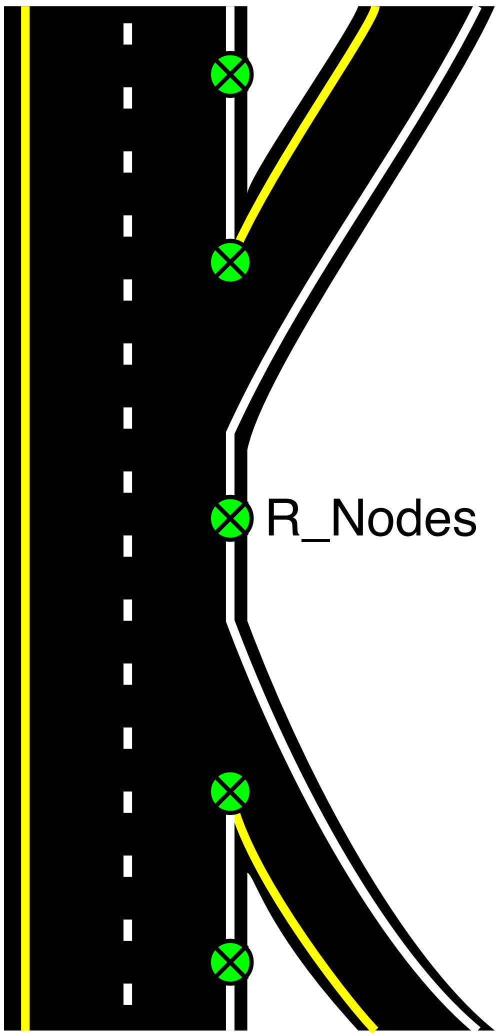
Generating tiles takes about 2 hours on a 16 processor system, or about 13K tiles/min.

```
su - tms
cd ~/osm/mapnik
export MAPNIK_MAP_FILE=~/osm/mapnik/iris_osm.xml
export MAPNIK_TILE_DIR=/home/tms/osm/tiles
# Customize osm.xml
./generate_xml.py  --dbname gis --accept-none
vim generate_tiles_multiprocess.py
        NUM_THREADS = number of processors in your
        bbox = (-97.5, 43.0, -89.0, 49.5)
        minZoom = 6
        maxZoom = 16
        # comment all of the european cities liste
timer ./generate_tiles_multiprocess.py
# Copy tiles to server hosting them
scp -rp ~/osm/tiles/* root@tilehost:/var/www/html/
```

3.5.11 Create Map Extents in IRIS database

In the IRIS client, select the `View ➔ System ➔ Map Extents` menu item to bring up the Map Extents form. This form enables the creation of map extents, which are buttons allowing quick selection of a map location and zoom level. Each map extent will be displayed as a button on the upper right area of the map (next to the zoom in/out buttons). First, a "Home" extent should be created, which is selected by default when a user logs in. Other extents can also be created for commonly selected map spots.

3.6 Road Topology



For many of the features IRIS provides, a model of the road network topology is required. This model consists of a set of roadway nodes, or **r_node**s, which describe all the points of interest on the road network. For instance, each entrance or exit ramp, detector station or intersection would be represented by an r_node.

All r_nodes with the same road and direction-of-travel are grouped into **corridors**. The r_nodes within a corridor are automatically sorted based on their coordinates. Corridors are linked to each other by exit and entrance r_nodes. For example an exit r_node from Northbound road A to Eastbound road B would automatically link to an entrance onto Eastbound road B from Northbound road A.

3.6.1 Road Setup

Before creating the r_nodes, some roads must be created. Select the `View ➔ System ➔ Roads` menu item to bring up the Roads form. To create a new road, select the last (empty) cell in the Road column and type the name of the road. This name can be up to 20 characters in length, and uniquely identifies the road. It's a good idea to capitalize each word, and abbreviate any designators — for example, "Main St".

The next column is **Abbrev**. Here, an abbreviation of up to 6 characters can be entered. This is used for labeling detectors and stations.

The **Road Class** determines the presentation of the road on the map. For example, a road marked "Freeway" would be drawn thicker and more prominent than an "Expressway" or "Arterial".

Direction and **Alt Dir** determine the primary and secondary compass directions of the road.

3.6.2 R_Node Location

Select the **R_Nodes** tab on the client interface to create or edit r_nodes. In the **Selected Roadway Corridor** frame, there is an **Add** button. Press the button to start creating a new r_node. The mouse cursor will change to a **target** while over the map. Select the point on the map where the r_node should be located and left-click the mouse. The new r_node will appear on the map, and in the list for the selected corridor.

In the **Selected Roadway Corridor** frame, there is a **Corridor** widget containing a list of all corridors which contain at least one r_node. When a corridor is selected, the r_node list will be populated with all r_nodes on the corridor. Also, any new r_nodes created will be on the selected corridor.

Once an r_node is selected, any fields on the **Location** tab can be edited. Changing the roadway or direction will move the r_node to another corridor. Other attributes are nearest cross-street, cross-street direction, and notes.

3.6.3 R_Node Setup

The r_node **Setup** tab contains several more attributes.

node_type

There are six different r_node types:

Code	Node Type	Description
0	station	Mainline detector station or shape node
1	entrance	Entrance ramp onto corridor
2	exit	Exit ramp off corridor
3	intersection	At-grade intersection with another corridor
4	access	Network access (for traveler information)
5	interchange	Freeway interchange (for traveler information)

transition

An r_node transition describes how entrance or exit nodes connect with linked nodes.

Code	Transition	Description
0	none	No transition — non-entrance or -exit r_nodes
1	loop	Interchange loop ramp — "leaf" part of cloverleaf
2	leg	Typical "leg" ramp
3	slipramp	Transition ramp between parallel corridors
4	CD	Transition to/from collector/distributor road
5	HOV	Bypass ramp for high-occupancy-vehicles
6	common	Common section joining two corridors
7	flyover	Flyover ramp (bridge over mainline)

lanes
Number of lanes at r_node. For a ramp node, this is the number of lanes entering or exiting the mainline.

shift
Lane shift is the difference (number of lanes) between the corridor reference lane and the attach side of the r_node. This value is used for lane continuity analysis within a corridor. Each corridor has a reference lane, which corresponds to the leftmost lane along the entire corridor.

station_id
The station_id is a unique identifier for the detectors associated with the r_node. It is displayed on the traffic layer, and is also used for identifying travel time destinations.

speed_limit
This is the posted speed limit at the r_node. The units are miles per hour, but should in future be dependent on locale (KPH). For ramp r_nodes, it may be used to indicate the advisory speed limit on the ramp. The limit is used in travel time estimation.

pickable
For traveler information systems, an r_node can be marked pickable if a user is allowed to pick a node as an origin or destination. NOTE: This feature has not been implemented yet.

above
When rendering the traffic layer, r_nodes are rendered in two passes. The second pass consists of r_nodes which have been marked **above**. This prevents overpasses from being drawn incorrectly.

attach_side
This flag indicates whether the r_node is attached to the left side of the road. This can be used to create left entrance or exit ramps.

active
This flag allows an r_node to be deactivated. An inactive r_node will not be included in any corridor or drawn on the traffic layer.

3.6.4 R_Node Detectors

The r_node **Detectors** tab allows vehicle detectors to be created and associated with an r_node.

3.7 Comm Links

A **comm link** is a network connection to an external device or system. IRIS is capable of supporting hundreds of simultaneous comm links. The comm link interface is found on the `View ➔ Maintenance ➔ Comm Links` menu item. Each comm link has a number of attributes, including URI and protocol.

The URI includes a DNS host name or network IP address, and port number, using the standard "host:port" convention. The URI can also contain an optional "scheme" prefix, which can be either "udp://", "tcp://" or "modem://". If present, the scheme will override the default URI scheme for the selected protocol. For example, to use the Pelco D protocol over TCP (instead of the default UDP), add a "tcp://" scheme prefix to the URI.

The protocol determines what type of device or system is on the other end of the comm link. The following table contains the communication protocols supported by IRIS.

Protocol	Description	Default Scheme
NTCIP Class A	DMS control, ILCS (no HDLC)	UDP
NTCIP Class B	DMS control, ILCS over serial (HDLC)	TCP
NTCIP Class C	DMS control, ILCS (no HDLC)	TCP
DMS XML	DMS control to external system	TCP
Msg feed	DMS messages from external system	TCP / HTTP
MnDOT 170 4	VDS, metering, DLCS, beacons, alarms	TCP
MnDOT 170 5	VDS, metering, DLCS, beacons, alarms	TCP
Canoga	VDS (GTT Canoga Traffic Sensing System)	TCP
SS105	VDS (original Wavetronix SmartSensor)	TCP
SS125	VDS (Wavetronix SmartSensor HD)	TCP
RTMS G4	VDS (RTMS G4 sensor)	TCP
Pelco D	PTZ for Pelco cameras	UDP
Manchester	PTZ for AD cameras	UDP
Vicon	PTZ for Vicon cameras	UDP
Infinova	PTZ for Infinova cameras	TCP
Cohu	PTZ for Cohu cameras	TCP
STC	Gate Arm control for HySecurity SmartTouch Controllers	TCP
Pelco Switcher	Video switching for Pelco	TCP
Org 815	RWIS (ORG 815 sensor)	TCP
SSI	RWIS (SSI RWIS sensor)	TCP / HTTP
DLI DIN Relay	DLCS, beacons	TCP / HTTP

3.7.1 DMS Msg Feed Protocol

The "msg feed" protocol can be used to interface IRIS with an external system that generates DMS messages. Periodically, IRIS will poll the URI (using HTTP) for DMS messages. The external system should respond with an ASCII text file, with one line per message to be deployed.

Each line must contain 3 fields, separated by tab characters "\t" (ASCII 0x09), and terminated with a single newline character "\n" (ASCII 0x0A). The fields are DMS name, MULTI string, and expiration time. The DMS name must exactly match one of the DMS as identified by IRIS. The MULTI string specifies a message to display on the sign, using the MULTI markup language, from NTCIP 1203. For example, "ROAD CLOSED[nl]AHEAD[nl]LEFT LANE CLOSED" is a valid three-line MULTI message. The expiration field indicates the date and time that the message should expire. It is in "yyyy-MM-dd HH:mm:ssZ" format, which includes date, time and time zone. A valid expiration time would be 2012-05-15 11:37:00-0600.

There are a couple of other required steps to enable a message feed to operate. First, an action plan must be created which associates a DMS action with the message feed. The DMS action must be associated with a quick message that references the message feed, using a special MULTI tag. So, if the message feed is on a Comm Link called "LFEED", then the quick message MULTI string must be "[feedLFEED]". Also, the action plan must be active and deployed. This requirement allows only administrator-approved DMS to be controlled by the message feed.

The other requirement for message feeds is that all messages used by the feed must be defined in the DMS message library. This requirement allows only administrator-approved messages to be deployed by the message feed. In some circumstances, it may be appropriate to disable this checking. For example, if the message feed host is fully trusted and there is no possibility of man-in-the-middle attacks between IRIS and the feed host. In this case the "msg_feed_verify" system attribute can be set to "false" to disable this check.

3.7.2 DIN Relay

DIN relay is is product by DLI which allows simple devices to be turned on of off through an HTTP (web) interface. Each device can control up to 8 outlets, which for a DLCS means 2 lanes with 3 indications each.

3.8 Controllers

After a comm link has been created, a **controller** must be associated with it. Some protocols support **multi-drop** addressing, and others are **single-drop** only. Multi-drop addressing allows multiple controllers to share the same comm link. Each controller is assigned a unique (to the comm link) drop address, which is used to route all messages sent to the controller. Controllers can be assigned to a **cabinet**, which has location information, to allow displaying on a map.

3.8.1 I/O Pins

Each controller has a set of **I/O pins**, to which devices can be assigned. The function of these pins is protocol specific — see the following table.

Device Type	Protocol	I/O Pin(s)
DMS, ILCS	NTCIP (A, B, C), DMS XML	1
DLCS	MnDOT 170 (4, 5)	19 - 36
	DIN Relay	1 - 8
Camera	Pelco D, Manchester, Vicon, Infinova, Cohu	1
VDS	MnDOT 170 (4, 5)	39 - 62
	Canoga	1 - 4
	SS105, SS125	1 - 8
	RTMS G4	1 - 12
Ramp Meter	MnDOT 170 (4, 5)	2 or 3
Gate Arm	STC	1
Beacon	MnDOT 170 (4, 5)	2
	DIN Relay	1 - 8
RWIS	Org 815, SSI	1
Video Switcher	Pelco	1

3.9 Fonts

A font is a set of bitmapped glyphs for displaying text on a dynamic message sign (DMS). Each font can contain any of the printable characters in the ASCII character set (0x20 to 0x7E).

When selecting a font, a few parameters must be considered, such as pixel pitch, desired character height, and font weight. Pixel pitch can vary from 66 mm down to 20 mm or smaller. Typically, for freeway signs, characters should be between 350 and 400 mm (about 14 or 15 inches) high. See the table below for character heights based on these common sizes.

Font Height	Pixel Pitch			
	20 mm	33 mm	50 mm	66 mm
7 px	120 mm	198 mm	300 mm	396 mm
8 px	140 mm	231 mm	350 mm	462 mm
10 px	180 mm	297 mm	450 mm	594 mm
12 px	220 mm	363 mm	550 mm	726 mm
14 px	260 mm	429 mm	650 mm	858 mm
16 px	300 mm	495 mm	750 mm	990 mm
18 px	340 mm	561 mm	850 mm	1122 mm
20 px	380 mm	627 mm	950 mm	1254 mm

In the past, DMS have been used with upper-case only fonts, but with smaller pixel pitch, it is possible to create legible fonts which also include lower-case characters. If a message containing lower-case characters is used with an upper-case only font, IRIS will replace those characters with their equivalent.

Fifteen fonts are included in the IRIS template SQL script. These fonts are designed to have a similar visual style.

Number	Font Name	Description
1	07_char	7x5 character matrix
2	07_line	7 pixel high line-matrix
3	08_full	8 pixel high full-matrix
4	09_full	9 pixel high full-matrix
5	10_full	10 pixel high full-matrix
6	11_full	11 pixel high full-matrix
7	12_full	12 pixel high full-matrix
8	12_full_bold	12 pixel high full-matrix bold
9	13_full	13 pixel high full-matrix
11	14_full	14 pixel high full-matrix
13	16_full	16 pixel high full-matrix
14	18_full	18 pixel high full-matrix
15	20_full	20 pixel high full-matrix (numerals only)
16	24_full	24 pixel high full-matrix (numerals only)
17	_09_full_12	9 pixel high (12 with lower case descenders)

The IRIS client also contains a font editor which can be used to design new DMS fonts. The font interface is found on the `View ➔ Message Signs ➔ DMS Fonts` menu item.

4 Devices

A device is a hardware field control or sensing system. IRIS supports several different device types: dynamic message signs, cameras, vehicle detection, lane-use control signs, ramp meters, gate arms, beacons, alarms, road weather information systems, and lane marking.

For any device to be activated, it must first be connected to one of the I/O pins on a controller. The controller must also be associated with a comm link which communicates using the appropriate protocol for the device.

4.1 Dynamic Message Signs (DMS)

A dynamic message sign (DMS) is a sign which is capable of changing the message displayed to motorists. They can be classified as character-matrix, line-matrix and full-matrix, depending on the spacing between pixels. Some are monochrome, and some support full color display. All of these configurations are supported by IRIS.

The following operations can be performed on a DMS:

- Querying currently displayed message
- Displaying new messages
- Querying configuration information
- Querying diagnostic information
- Sending fonts and graphics

4.1.1 DMS Setup

FIXME

4.1.2 MULTI (Markup Language for Transportation Information)

DMS messages are specified using **MULTI**, as defined by the NTCIP 1203 standard. In MULTI, messages are simple ASCII strings, with formatting or other instructions denoted by tags inside square brackets. For example, the tag `[nl]` indicates a new line. Most of the useful standard MULTI tags are supported, as well as a few which are specific to IRIS. The following table summarizes the tags supported by IRIS.

Tag	Description	Supported
[cb*x*]	Message background color	Yes
[pb*z*]	Page background color	Yes
[pb*r*,*g*,*b*]		
[cf*x*]	Foreground color	Yes
[cf*r*,*g*,*b*]		
[cr*x*,*y*,*w*,*h*,*z*]	Color rectangle	Yes
[cr*x*,*y*,*w*,*h*,*r*,*g*,*b*]		
[f*x*,*y*]	Field data	No
[flt*x*o*y*]	Flashing text	No
[flo*y*t*x*]		
[fo*x*]	Change font	Yes
[fo*x*,*cccc*]		
[g*n*]	Place graphic	Yes
[g*n*,*x*,*y*]		
[g*n*,*x*,*y*,*cccc*]		
[hc*n*]	Hexadecimal character	No
[jl*n*]	Line justification	Yes
[jp*n*]	Page justification	Yes
[ms*x*,*y*]	Manufacturer specific	No
[mv...]	Moving text	No
[nl]	New line	Yes
[nl*s*]		
[np]	New page	Yes
[pt*n*]	Page time	Yes
[pt*n*o*f*]		
[sc*x*]	Character spacing	Yes
[tr*x*,*y*,*w*,*h*]	Text rectangle	Yes
[tt*n*]	Travel time	IRIS-only
[vsa]	Speed advisory	IRIS-only
[slow*s*,*b*,*u*,*dist*]	Slow traffic warning	IRIS-only
[feed*n*]	Message feed	IRIS-only

4.2 Cameras

Closed-circuit television cameras are commonly used to monitor traffic conditions. These cameras can transmit video to a traffic management center using either an analog signal (with dedicated cabling) or a digital stream (using an IP network). There are advantages to both approaches. Analog video has high picture quality and very low latency, while digital video is much cheaper and easier to route around.

To display camera video in IRIS, it must be digitally encoded.

4.2.1 Pan / Tilt / Zoom

Some cameras have a built-in pan/tilt/zoom (PTZ) functionality. IRIS can perform the following PTZ operations:

- Panning / tilting the camera
- Zooming in or out
- Recall preset positions
- Store preset positions

4.2.2 Video Switching

IRIS can also be configured to send commands to a video switcher system. This allows camera video (or streams) to be routed to a number of video monitors. Each monitor can be assigned to display the video from any specific camera. IRIS allows an operator to select one video monitor to be linked with their camera selection on the user interface. This way, every time the user selects a camera, it's video is immediately displayed on the associated monitor.

4.3 Vehicle Detection Systems

There are several types of **vehicle detection systems** (VDS) — also called **traffic sensing systems**. The oldest of these is the inductive loop detector, which is a wire placed under the surface of the road. It is able to magnetically detect the metal in vehicles as they pass over. Another type of sensor uses radar mounted on the side of the road. This type is **non-intrusive**, since the road surface does not need to be cut to install the detector. A third type – also non-intrusive – is video detection. These systems use video cameras and computer vision software to detect vehicles.

4.3.1 VDS Configuration

To create a detector, first select the r_node at the location of the detector. Then, select the **Detectors** tab for the r_node. Type the detector name in the text entry widget just above the **Create** button, and press that button. After selecting the new detector in the r_node detector table, its properties, such as **lane type** and **lane #** can be changed. In IRIS, lanes are numbered from right-to-left, starting with the right lane as 1.

4.3.2 VDS Protocols

IRIS supports several different protocols for communicating with vehicle detection systems. For most VDS protocols, traffic data is binned at a fixed time interval. Some protocols allow the binning interval to be adjusted. IRIS collects traffic data including **volume** (traffic counts), **occupancy**, **speed**, and **vehicle classification**. Some protocols do not use binning at all — instead they record timestamps and headway for each vehicle passing by the sensor. The following table summarizes features of each protocol.

Protocol	Addressing	Binning	Speed
MnDOT 170 4	4-bit (1-15)	30 seconds (+5 minute)	No
MnDOT 170 5	5-bit (1-31)	30 seconds (+5 minute)	No
Canoga	8-bit (1-255)	No (vehicle logging)	Double loops only
SS105	4-digit (0001-9999)	5 seconds to 1 month	Yes
SS125	32-bit (0-65535)	???	Yes
RTMS G4	32-bit (0-65535)	10 seconds to 1 hour	Yes

4.3.3 VDS Data Archiving

Every 30 seconds, an XML file is generated containing the most recent sample data from all defined detectors. The file is called det_sample.xml.gz, and it is written to the XML output directory

Sample data is archived only if the `sample_archive_enable` system attribute is set to `true`. Sample archive files are stored in `/var/lib/iris/traffic`, in a directory with the district ID name.

4.4 Lane-use Control Signs (LCS)

A lane-use control sign (LCS) is a sign which is mounted over a single lane of traffic (typically one for each lane). It can display a set of indications which either permit or restrict use of that lane.

4.4.1 "Intelligent" Lane-use Control Signs (ILCS)

An "intelligent" LCS is a full-matrix color DMS which is mounted directly over a lane of traffic. In addition to being used as an LCS, it can display variable speed advisories, or any operator-defined message as a regular DMS.

4.4.2 "Dumb" Lane-use Control Signs (DLCS)

A "dumb" LCS device is much simpler than an ILCS. It can display one of a fixed set of indications — typically a *green arrow* ↓ , a *yellow arrow* ↓ or a *red X* X). Each indication must be assigned to a separate I/O pin on a controller, as well as the DMS which represents the LCS.

4.5 Ramp Meters

A ramp meter is a traffic signal at an on-ramp which controls the rate of vehicles entering a freeway. Typically, one vehicle is permitted to enter for each green indication displayed on the signal.

The following operations can be performed on a ramp meter:

- Activating and deactivating meter
- Adjusting meter release rate
- Querying meter status
- Querying green counts
- Synchronizing clock
- Configuring time-of-day operation

4.6 Gate Arms

Gate Arms are traffic control devices which restrict access to a section of roadway. They can be used, for example, for reversible lanes which can be changed by time of day.

4.7 Beacons

A beacon is a light or set of lights that flashes toward oncoming traffic. Its purpose is to draw attention to a DMS or static sign. Sometimes, they are called **flashers** or **wig-wags**.

4.8 Alarms

An alarm is a simple device which is triggered when an event happens. The event could be an equipment failure, high temperature, etc. Alarms can be monitored to help maintain field devices.

4.9 Road Weather Information Systems

Road Weather Information Systems (RWIS) can sense data such as precipitation rate, road surface temperature, wind speed, etc.

4.10 Lane Marking

A lane marking device is also called **in-road lighting**. It allows the lane striping to be changed dynamically, by turning on and off lights embedded in the road.

5 Features

5.1 Traffic Map Layer

The IRIS client user interface includes a traffic map layer which is created automatically from the road topology. By default, this "segment" layer uses traffic **density** to determine the color of each segment in the layer. Other themes are available for **speed** and **flow**. The **Legend** menu at the top of the map can be used to view the thresholds used for each color in a theme.

Every 30 seconds, the client will make an HTTP request for the current traffic sample data XML file. The URL to locate that file is declared as a property in the /etc/iris/iris-client.properties file (on the IRIS server). The property is tdxml.detector.url, and it should point to the location of det_sample.xml.gz, as made available by apache on the IRIS server.

The appearance of the traffic map layer changes depending on the current map zoom level. If the zoom level is below 10, the layer will not be visible. At zoom levels 10 through 13, the layer will display segments as aggregate of all detectors in each mainline station. At zoom level 14 or above, each mainline detector will be displayed as a separate segment.

5.2 Incident Management
FIXME
5.3 Action Plans
FIXME
5.4 Date/Time Schedules
FIXME
5.5 Ramp Metering Strategies
FIXME
5.6 Travel Time Estimation
FIXME
5.7 Variable Speed Advisory
FIXME
6 Troubleshooting
6.1 Error Logs
FIXME
6.2 XML Output

There are a number of XML files which are written by the IRIS server periodically. These files contain configuration information about the system as well as realtime data concerning the current state. These files are stored in the /var/www/html/iris_xml/ directory.

Filename	Description	Period
tms_config.xml.gz	Configuration data for IRIS system. If the district server property is changed, the filename will be changed to match {district}_config.xml.gz.	24 hours
det_sample.xml.gz	Sample data from vehicle detection systems	30 seconds
stat_sample.xml.gz	Mainline station data from vehicle detection systems	30 seconds
incident.xml.gz	Current incident information	30 seconds
sign_message.xml.gz	Current DMS sign message information	30 seconds

6.3 Database Event Tables
FIXME
6.4 Debug Trace Logs
FIXME

7 Maintenance

Once an IRIS system is set up, there are a few maintenance tasks which need to be done to ensure reliable operation.

7.1 Database Backup & Restore

It is a good idea to backup the IRIS database on a regular basis. This can be done simply with a command such as the following:

```
pg_dump tms | gzip > tms-20120523.sql.gz
```

This command can be placed in a simple python or bash script which is run by cron once per day. For off-site backups, the file could be copied to another host, using scp or a similar utility.

To restore the tms database from one of these backup files, you will first need to shut down the IRIS server (and anything else connected to the tms database). Then, run the following commands (as tms user):

```
dropdb tms
createdb tms
zcat tms-20120523.sql.gz | pgsql tms
```

7.2 IRIS Upgrades

Upgrading an IRIS system to a new version is a simple process. There are two different scenarios to deal with when upgrading: **stable** versus **unstable**. Either way, it's advisable to backup the database before attempting an upgrade.

A stable upgrade is when the major and minor IRIS version numbers do not change — just the **micro** version. So, upgrading from 3.145.0 to 3.145.1 would be a stable updgrade. This type of upgrade never involves a database schema change. With the new rpm file available, a stable upgrade can be accomplished with the following commands (as root):

```
yum update iris-{major}.{minor}.{micro}-{build}.nc
iris_ctl update
systemctl restart iris.service
```

An unstable upgrade is when either the major or minor IRIS version numbers change. Upgrading from 3.144.0 to 3.145.0 would be an unstable upgrade. This type of upgrade always involves a database schema change. To upgrade the database schema, an SQL migrate script must be run for each intervening version. The following commands can be used to perform an unstable upgrade (as root):

```
yum update iris-{major}.{minor}.{micro}-{build}.nc
psql tms -f /var/lib/iris/sql/migrate-{major}.{mi
iris_ctl update
systemctl restart iris.service
```

Note: when skipping one or more minor versions in an upgrade, **all** migrate scripts must be run. So, when upgrading from 3.143.0 to 3.145.0, the database upgrade would consist of these commands:

```
psql tms -f /var/lib/iris/sql/migrate-3.144.sql
psql tms -f /var/lib/iris/sql/migrate-3.145.sql
```

Warning: there is no supported method of **downgrading** an IRIS system. If a downgrade is required, the database should be restored from a backup and the IRIS rpm should be reinstalled.

7.3 System Software Updates

FIXME

7.4 Monitoring Logs

FIXME

8 Development

8.1 Building

Install Fedora onto the development system. Some additional development packages must be installed.

```
yum install java-1.8.0-openjdk-devel ant-junit mer
```

In a suitable development directory, clone the iris mercurial repository.

```
hg clone http://iris.dot.state.mn.us/hg/index.cgi/
```

Create lib/ directory in iris development repository.

```
mkdir iris/lib/
```

Download the JavaMail API from Oracle. Unzip the JavaMail archive and locate the mail.jar file. Copy the mail.jar file to the lib/ directory in the iris repository.

Copy `iris-build.properties` from etc/ to lib/. Edit `lib/iris-build.properties` file as necessary for your organization.

Create the key for signing jar packages. This is required for Java Web Start, which is used by the IRIS client. The key should go into your default keystore (~/.keystore).

```
keytool -genkeypair -alias signing-key
```

Put the following lines into your ~/.ant.properties file.

```
debug=off
sign.store=${user.home}/.keystore
sign.store.pass=password
sign.alias=signing-key
sign.alias.pass=signing-password
hgbase.url=http://iris.dot.state.mn.us/hg/index.cg
```

Now, the IRIS rpm file can be built with the following commands. (The current directory should be the root of the iris repository.

```
ant rpm
```

If there are no errors, the new rpm file should be in the `build/rpm/RPMS/noarch/` directory.

8.2 History

March 1999
: Started work on NTCIP DMS control

May 2007
: First GPL open source release

May 2008
: IRIS live in Caltrans D10

April 2011
: Officially adopted by Caltrans

8.3 Future Plans

FIXME

8.4 Contributing

FIXME

8.5 Coding Style

The coding style for IRIS is focused on making code more readable. Readability is the most important factor for software maintenance, and it also helps new developers who are unfamiliar with the codebase.

First and foremost, it is recommended that each method fit in one page of text (30 lines by 80 columns). Whenever possible, longer methods should be decomposed to abide by this recommendation. This allows a developer to quickly read and understand the method's logic. It is much easier to spot and correct bugs in shorter methods. There should be no blank lines within a method.

With methods longer than one page, it can be beneficial to have a single exit point (return). But for shorter methods (always the goal), this is not so important, and can reduce readability. It doesn't make sense to add a local "ret" variable with a single return statement at the end when the whole method can be read at a glance.

Each block indent should be 8 columns (using tabs instead of spaces). This makes the blocks more obvious, and is easier to read than the 4 spaces commonly used in other Java projects. Always use an indented block with if, for, while, etc. statements. This makes it much easier to follow the control flow than when it is combined onto the same line as the statement. Don't put an opening curly brace on a line by itself unless the method declaration or loop statement spans multiple lines. For single line blocks (if, for, while), do not use curly braces.

Each method should have a javadoc comment describing its purpose. All parameters and return values should be documented as well. Comments within a method should be used sparingly — never describe what the code is doing, but instead **why** it was written in a certain way. Code which is "commented out" should **never** be committed to a repository.

Unary operators, such as ++ -- ! ~ should be adjacent to their operand with no space. Binary operators, such as + - / * % & | && || > = < >> == != << >>> >= <= ^ should be separated from their operands by a space.