iTETRIS Building, Installation and Configuration Guidelines

Table of Contents

Table Index

Figure Index

1 INTRODUCTION

The iTETRIS project [http://ict-itetris.eu/] implements an open-source integrated wireless and traffic simulation platform that allows testing the efficiency of cooperative traffic management strategies (from now on Applications) using V2X technologies over large scale scenarios.

The whole iTETRIS platform consists of four main building blocks as illustrated in the following Figure 1. SUMO and ns-3 are two open source platforms, respectively for traffic and wireless simulation, which are integrated by a third block called iTETRIS Control System (iCS). Application algorithms supporting traffic strategies can be independently implemented and run on the top of the iCS in the Applications block. Triggered by Applications' commands, ns-3 simulates V2X transmissions in vehicular wireless communications scenarios. Receptions deriving from these communications are notified to the Applications, which in consequence can produce actions to be undertaken in the road traffic scenario simulated by SUMO. As a result, SUMO continuously feeds the other blocks with vehicles' position updates, whose knowledge is essential for wireless simulations. As the central module in the architecture, the iCS facilitates the exchange of data between all the functional blocks and supports the simulation flow by control and synchronization functions.

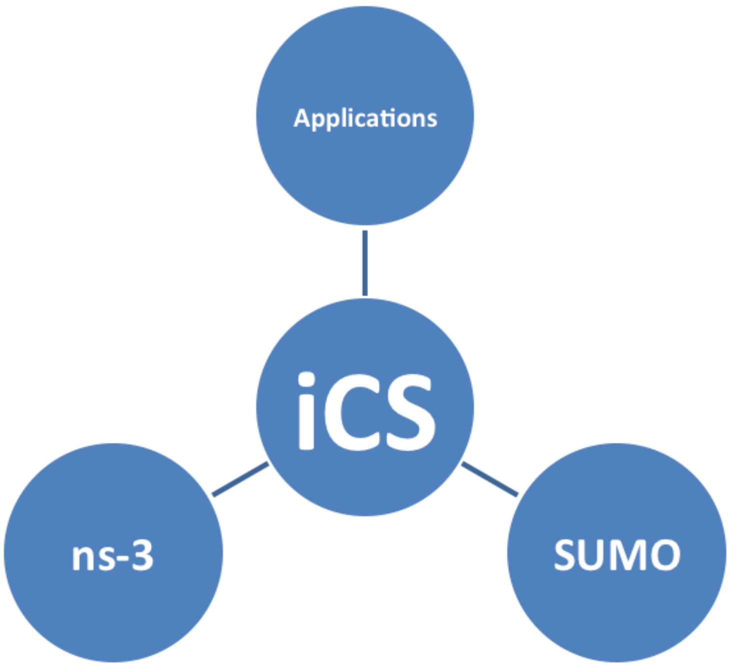


Figure 1. iTETRIS platform main building blocks relationship.

The following sections of this document are aimed to explain how to build and install the iTETRIS platform. A section explaining the configuration of iTETRIS simulations is also included. At the end of the document, a section describing how to run a sample Demo Application is presented along with some guidelines on how to couple new applications to iTETRIS.

The currently supported SUMO is **version is 0.14.0**. But please check regularly on the community website for update of its support.

SUM can be downloaded as in the official site [http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Main_Page].

2 BUILDING AND INSTALLING THE iCS

The iCS has been designed and developed to run under Ubuntu. The supported version is 10.10.

Requirements to build the iCS include:

- GNU autotools (autoconf, automake, libtool)
- g++ compiler

To install them, please enter the following commands in the terminal:

```
sudo apt-get install autoconf automake libtool
sudo apt-get install g++
```

Other requirements are:

- Xerces libraries
- Geographic libraries

Please find these libraries in the repository iCS/libs/ of the iTETRIS distribution. Uncompress the folders geographiclib-1.2.tar.gz and xerces-c-3.0.1.tar.gz, then compile and install them by entering the following commands from their respective folders:

```
sudo ./configure
sudo make
sudo make install
```

Having the requirements installed, from the iCS base folder enter the command that creates the necessary GNU “autotools” files:

```
sudo make -f Makefile.cvs
```

To configure how the way how iCS is built, enter the following command:

```
sudo ./configure --enable-sumo --enable-ns3 --enable-applications --with-geographic-libraries=/usr/local/lib --with-geographic-includes=/usr/local/include/GeographicLib --with-xerces-libraries=/usr/local/lib --with-xerces-includes=/usr/local/include/xercesc
```

If you need additional developing support, the following tags are available:

- --enable-log: compiles the iCS allowing to print logs messages according to the log levels (INFO, WARNING, ERROR.) given in the configuration file.
- --enable-debug: compile the iCS in debug-mode

Finally, to compile and install the iCS execute the command:

```
sudo make install
```

The binaries are installed in the “usr/local/bin” folder.

3 BUILDING AND INSTALLING ns-3

The iTETRIS platform currently supports the ns-3 platform in its **3.7 version** for wireless communications’ simulation. However, the openness and modularity of its source code allows for modifications in order for iTETRIS to be coupled with other open source wireless simulators. In the iTETRIS project, ns-3 has been developed to include a number of new functionalities. This evolved version of ns-3 is included in the iTETRIS distribution. To correctly run the iTETRIS ns-3 code, it is necessary to install XML libraries. The package containing them can be installed via “Synaptic”, in case of using Ubuntu, or via console by executing the following command:

```
sudo apt-get install libxml++2.6-dev
```

To configure, build and install the ns-3 source code, from the ns-3 folder please enter the following commands in the terminal:

```
sudo ./waf configure
sudo ./waf install
```

Finally, make the libns3.so available to iTETRIS:

```
sudo cp /usr/local/lib/libns3.so /usr/lib/.
```

4 BUILDING AND INSTALLING SUMO

Similarly to what said for the wireless simulator, iTETRIS has been design not to be initially tied to a particular traffic simulator. In the current release of iTETRIS, the SUMO traffic simulator in its version **0.14.0** is used. To correctly run SUMO, various libraries are required. Please get and install them by running the following commands:

```
sudo apt-get install libgdal1-dev proj libxerces-c2-dev
sudo apt-get install libfox-1.6-dev libgl1-mesa-dev libglu1-mesa-dev
```

Patch “libgdal” version (for Ubuntu 10.10)

```
cd %BASE-SUMO%
vim configure.ac
replace LIB_GDAL="gdal" to LIB_GDAL="gdal1.6.0"
```

Configuring SUMO for Ubuntu 10.10

```
sudo –f Makefile.cvs

sudo ./configure –with-fox-libraries=/usr/lib --with-fox-
includes=/usr/include/fox-1.6 --with-gdal-libraries=/usr/lib --with-
gdal-includes=/usr/include/gdal --with-proj-libraries=/usr --with-
geographic=/usr/local/lib --with-geographic-
includes=/usr/local/include/GeographicLib --with-xerces=/usr/local/lib
--with-xerces-includes=/usr/local/include/xercesc
```

Note: if the Makefile.cvs does not exist in your SUMO distribution, please use the following code instead:

```
autoreconf --force --install

sudo ./configure –with-fox-libraries=/usr/lib --with-fox-
includes=/usr/include/fox-1.6 --with-gdal-libraries=/usr/lib --with-
gdal-includes=/usr/include/gdal --with-proj-libraries=/usr --with-
geographic=/usr/local/lib --with-geographic-
includes=/usr/local/include/GeographicLib --with-xerces=/usr/local/lib
--with-xerces-includes=/usr/local/include/xercesc
```

Then, to build and install SUMO, from its base folder execute the following commands:

```
sudo make
sudo make install
```

5 iTETRIS SIMULATION CONFIGURATION

In this section, a description of how an iTETRIS simulation can be configured is given. The configuration is based on a hierarchy of XML formatted files. Every block composing the iTETRIS platform can be configured using dedicated configuration files. The set and organisation of these files is presented in Figure 2.

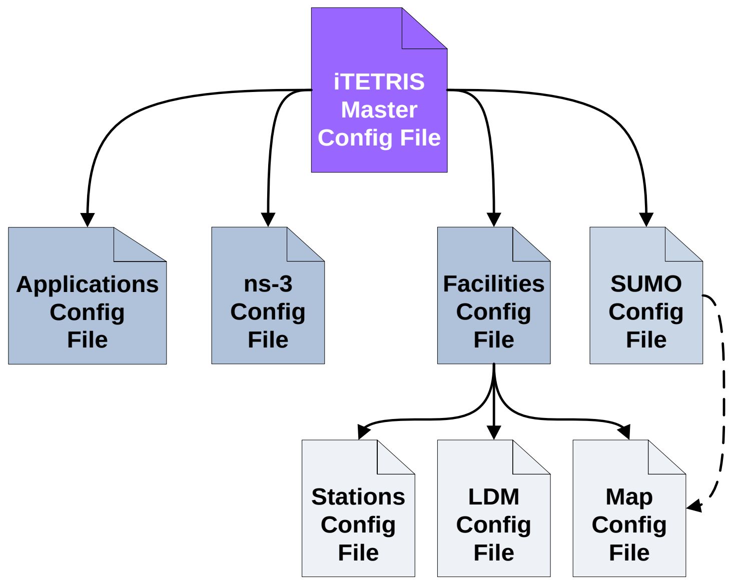


Figure 2. iTETRIS configuration files schema.

The description of the SUMO configuration file is out of the scope of this document, as its format can be found in the SUMO project official site [http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Main_Page]. The configuration files for the rest of the iTETRIS blocks are described in the following subsections.

5.1 iTETRIS Master Configuration File

Hierarchically organized, the iTETRIS configuration file is the entry point to the customization of the simulation. An example of the iTETRIS master configuration XML file is shown in the following:

```
<configuration>
	<scenario>
		<begin value="0"/>
		<end value="3000"/>
		<penetration-rate value="100"/>
		<facilities-config-file value="facilities-config-file.xml"/>
		<message-reception-window value="5"/>
		<interactive value="false"/>
	</scenario>
	<trafficsim>
		<traffic-executable value="sumo-gui -c"/>
		<traffic-file value="./TestNetwork/droomdorp.traci.sumo.cfg"/>
		<traffic-host value="localhost"/>
		<traffic-port value="5500"/>
	</trafficsim>
	<communicationsim>
		<communication-executable value="main-inci5"/>
		<communication-host value="localhost"/>
		<communication-port value="1982"/>
		<communication-general-params-file value="configGeneral.xml"/>
		<communication-config-technologies-file value="configTechnologies-
ics.xml"/>
	</communicationsim>
	<applications>
		<app-config-file value="application-config-file.xml"/>
	</applications>
	<logs>
```

```
                <ics-log-path value="ics-log.txt"/>
                <ics-log-level value="INFO"/>
                <ics-log-time-size value="100"/>
                <ns3-log-path value="ns3-log.txt"/>
        </logs>
</configuration>
```

As it can be seen, it is divided in four main blocks:

- Simulation basic information set.
- Traffic simulator configuration, SUMO in iTETRIS.
- Wireless communication simulator configuration, ns-3 in iTETRIS.
- Traffic management applications configuration file path.
- Logging configuration.

The first block indicated by the tag <*scenario*> (Table 1) is related with the main parameters of the scenario, here the most important values is the second in which the simulation ends.

TAG NAME	DESCRIPTION
Begin	Time step in which the Applications will go active.
End	Time step in which the Applications will deactivate.
Penetration-rate	Percentage of the total amount of vehicles equipped with a radio communication device.
facilities-config-file	Path of the file in which the configuration of the iCS Facilities is defined.
message-reception-window	Defines how many time steps the iCS will store radio messages that have been scheduled for transmission in the wireless simulator. After a number of simulation time steps equal to message-reception-window the iCS discards them considering that those messages did not reach their destination in the wireless simulation environment.
Interactive	Interactive tag allows running the simulation stopping the simulation in each time step. To make the simulation continue press the ENTER key

Table 1. Simulation basic information set tag definition.

```
<scenario>
        <begin value="0"/>
        <end value="3600"/>
        <penetration-rate value="100"/>
        <facilities-config-file value="facilities-config-file.xml"/>
        <message-reception-window value="5"/>
        <interactive value="false"/>
</scenario>
```

The next block <*trafficsim*> refers to data to be provided to the iCS for it to setup and establish the connection with SUMO (Table 2).

TAG NAME	DESCRIPTION
traffic-executable	Traffic simulator executable name, including parameters. For example, if the sumo-gui is installed, by setting this value to “sumo-gui -c” the traffic simulation will execute in a graphical user interface showing the road network scenario and the vehicles driving over it.
traffic-file	Traffic simulator configuration file.
traffic-host	IP address where SUMO is listening for commands.
traffic-port	Port number where SUMO is listening for commands.

Table 2. Traffic simulator configuration tag set.

```
<trafficsim>
        <traffic-executable value="sumo-gui -c"/>
        <traffic-file value="./TestNetwork/droomdorp.traci.sumo.cfg "/>
        <traffic-host value="localhost"/>
        <traffic-port value="1984"/>
</trafficsim>
```

Following the same approach, the block <*communicationssim*> specifies the needed data for the iCS connect with ns-3 simulator (Table 3).

TAG NAME	DESCRIPTION
communication-executable	Name of the wireless communications simulator executable.
communication-host	IP address where the communications simulator is listening for commands.
communication-port	Port number where the communications simulator is listening for commands.
communication-general-params-file	File (.txt) providing general communications simulation attributes. This file can be created through the ns-3 functionality “ConfigStore”.
communication-config-technologies-file	File (.xml) specifying the “installers” of the radio communications devices to be simulated in the wireless communications simulator. An installer allows installing and configuring a given radio communications device (i.e. ITS G5A, WiMAX, etc.) on a simulated node everytime the iCS requests its creation.

Table 3. Wireless communications simulator configuration tag set.

```
<communicationsim>
        <communication-executable value="main-inci5"/>
        <communication-host value="localhost"/>
        <communication-port value="1982"/>
        <communication-general-params-file value="configGeneral.xml"/>
        <communication-config-technologies-file value="configTechnologies-ics.xml"/>
</communicationsim>
```

The traffic management applications are specified in the main file by a dedicate tag *<applications>*. The path of the XML configuration file devoted to store the necessary values to locate the applications in the operating system, launch and connect the iCS to them is indicated here.

TAG NAME	DESCRIPTION
app-config-file	Path of the file specifying the necessary values to configure the Application.

Table 4. Application configuration file address tag.

```
<applications>
      <app-config-file value="application-config-file.xml"/>
</applications>
```

To close the iTETRIS master configuration file one more block *<logs>* is defined to specify the path of the files in which the iCS writes logging values. Differently from the other blocks, this block is optional and can be omitted (Table 5).

TAG NAME	DESCRIPTION
ics-log-path	The iCS logs file location and name.
ics-log-level	Specifies how the iCS logs file message. Options: INFO, WARNING, ERROR.
ics-log-time-size	To control the size of the logging files. Defines the interval of time of the logging messages generated per file.
ns3-log-path	The ns-3 logs file location and name. It Is delivered as an argument to ns-3.

Table 5. Logging configuration tag set.

```
<logs>
      <ics-log-path value="ics-log.txt"/>
      <ics-log-level value="INFO"/>
      <ics-log-time-size value="100"/>
      <ns3-log-path value="ns3-log.txt"/>
</logs>
```

It is important to highlight that the iTETRIS platform offers the possibility to create the iTETRIS XML master file from the command line following the format of the example below:

```
iCS -end 10 -penetration-rate 100 \
--facilities-config-file facilities-config-file.xml \
--traffic-host localhost \
--traffic-port 1984 \
--traffic-file ./TestNetwork/droomdorp.traci.sumo.cfg \
--communication-file ns3-config.txt \
--communication-host localhost \
--communication-port 1982 \
--ics-log-path ics-log.txt \
--ns3-log-path ns3-log.txt \
--apps /home/user/application-config-file.xml \
--ics-log-path ics-log.txt \
--ns3-log-path ns3-log.txt \
--save-configuration itetris-config-file.xml
```

5.2 iCS Facilities Configuration

iTETRIS is aligned with the communication architecture defined by the ETSI Technical Committee for Intelligent Transport Systems (ETSI TC ITS). Part of the architecture is the Facilities layer. The components of the layers are divided between ns-3 and iCS in order to improve the performance of the platform. This section describes how the Facilities in the iCS are configured.

The iCS Facilities configuration file complements the data of the iTETRIS master file described in the previous chapter. The file contains the paths and the name of the files for the configuration of the iCS Facilities block and the coordinates of the origin point of the map.

5.2.1 Facilities Configuration File

```
<facilities>
  <localCoordinates latitude="44.494218" longitude="11.346486" altitude="0.0"/>
  <mapConfig mapConFilename="./TestNetwork/droomdorp.net.xml" />
  <stationsConfig stationsConFilename="stations-config-file.xml" />
  <LDMrulesConfig LDMrulesConFilename="LDMrules-config-file.xml" />
</facilities>
```

TAG NAME	DESCRIPTION
localCoordinates	The Origin Geodetic Coordinate of the map. It is used to position the Cartesian Origin Point (0,0,0) on the map, located usually at the lowest left corner. This must be coherent with the map indicated in the mapConfig.
mapConfig	The location of description of the road map. It MUST be identical to the file used by the traffic simulator (indicated in the .cfg file for the traffic simulator.
stationConfig	The location of the ITS station description file.
LDMrulesConfig	The location of the LDM rule description file.

Table 6: Description of the Facilities Configuration Parameters

5.2.2 Map Configuration File

The map configuration file is an XML file that contains the description of the topology map. In the case where the traffic simulator is SUMO, the map file MUST be the net.xml file that describes the road topology for the SUMO simulation.

5.2.3 Stations Configuration File

In the following example the stations configuration file is presented.

```
<stations>
    <default>
        <RATseed value="12345" />
        <mobileStas>
            <mobileSta RAT-type="0" penetration-rate="100"   communication-
profile="WaveVehicle"/>  <!-- IEEE 802.11p/ETSI ITS GS -->
            <mobileSta RAT-type="1" penetration-rate="80"  communication-
profile="UmtsVehicle"/>  <!-- UMTS -->
            <mobileSta RAT-type="2" penetration-rate="20"  communication-
profile="WimaxVehicle"/> <!-- WiMAX -->
             <mobileSta RAT-type="3" penetration-rate="10" communication-
profile="DvbVehicle"/>   <!-- DVB-H -->
        </mobileStas>

        <fixedStas> <!-- Each Fixed Station has only one RAT available -->
            <fixedSta id="1" x="0" y="0" RAT-type="0" enabledRAT="0" communication-
profile="WaveRsu"/>
        </fixedStas>
    </default>
</stations>
```

In the example given above we can see that the file is divided in two parts. In the first part, information about the radio access technologies is given, while in the second part the fixed stations are declared

More specifically, the part about the mobile station contains the seed value of the random generator used for defining which technologies every mobile station entering the simulation area is equipped with. The default penetration rate of each radio access technology is defined below the random seed (in the above shown example, 80% of the vehicles created by the iCS will be equipped with UMTS radio technology, while 100% of them will be equipped with IEEE 802.11p/ETSI ITS G5 devices). Note that also a communication-profile field is given. In this field it is specified, if necessary, the specific profile that the communication simulator (e.g., ns-3) will use for the mobile stations. In addition, if more than one profile is defined for a certain technology, this has to be considered as different technology for the iCS. That more specifically means that the “enum” variable “RATID” in the “iCS/src/utils/ics/iCStypes.h” file must contain additional entries than the basic {WAVE, UMTS, WiMAX, DVBH}.

The part about the fixed radio stations specifies their ID, position (either as local x-y coordinates or as latitude and longitude) and radio access technology used and whether this is enabled at the beginning of the simulation (value “1” indicates enabled, while “0” indicates disabled). To create an RSU based on IEEE 802.11p/ETSI ITS G5 radio technology the value “WaveRsu” as to be used. For UMTS nodes B, WiMAX and DVB base stations the values “UmtsBs”, “WimaxBs” and “DvbhBs” have to be used, respectively.

5.2.4 LDM Configuration File

Finally, the LDM-rules-configfile.xml contains necessary parameters for the configuration of the LDM logic implemented inside the iCS Facilities. The skeleton of the XML file is provided.

```
<LDMrules>

    <defaultMessageLifeInterval value="" /> <!-- Value expressed in simulation steps --
```

```
>
    <relevantStationTypes fixed="" mobile="" /> <!-- 1=TRUE, 0=FALSE -->
    <relevantMessages cam="" denm=""/> <!-- 1=TRUE, 0=FALSE -->

    <relevantArea>
        <circle centerX="" centerY="" [centerLat="" centerLon=""] radius=""/>
        <ellipse focus1X="" focus1Y="" [focus1Lat="" focus1Lon=""] focus2X=""
focus2Y="" [focus2Lat="" focus2Lon=""] eccentricity=""/>
        <ellipse centerX="" centerY="" [centerLat="" centerLon=""] majorAxis=""
minorAxis="" rotationAngleRadians="" />

        <rectangle vertexAX="" vertexAY="" [vertexALat="" vertexALon=""] vertexBX=""
vertexBY="" [vertexBLat="" vertexBLon=""] vertexCX="" vertexCY="" [vertexCLat=""
vertexCLon=""] vertexDX="" vertexDY="" [vertexDLat="" vertexDLon=""] />

        <rectangle centerX="" centerY="" [centerLat="" centerLon=""] pointAX=""
pointAY="" [pointALat="" pointALon=""] pointBX="" pointBY="" [pointBLat=""
pointBLon=""] />

        <rectangle pointAX="" pointAY="" [pointALat="" pointALon=""] pointBX=""
pointBY="" [pointBLat="" pointBLon=""] height="" />

        <convexPolygon>
            <vertex X="" Y="" [Lat="" Lon=""] />
            <vertex X="" Y="" [Lat="" Lon=""] />
            <vertex X="" Y="" [Lat="" Lon=""] />
            <vertex X="" Y="" [Lat="" Lon=""] />
            <vertex X="" Y="" [Lat="" Lon=""] />
        </convexPolygon>

         <lanes>
             <lane ID="">
         </lanes>

         <edges>
             <edge ID="">
         </edges>

         <junctions>
             <junction ID="">
         </junctions>

    </relevantArea>

</LDMrules>
```

In the XML configuration file the only mandatory field is the defaultMessageLifeInterval. This value defines the number of simulation steps before deleting a message from the facilities tables when the expiry time is not explicitly defined (e.g. in CAM messages it is not defined, while in DENM messages it is).

The other fields in the XML are optional and define the rules for the relevance of the received messages. If a message is not relevant, it is not stored in the facilities tables. The rules accounts the receiver station type (a receiver is considered relevant only if it is of a certain type), message type (fixed stations or mobile stations or both are considered as relevant receivers) and relevance area (only receivers within a defined area are considered as relevant).

The relevance area, if set can be specified in different and various ways, as indicated in the example above. Note that the overall relevance area is the union of the areas indicated in the file. These areas can be either geometrically defined (please note that the points can be defined as x-y coordinates or as latitude and longitude) or road elements such as road lanes, road edges (i.e. streets segments) or junctions.

5.3 iTETRIS Applications Configuration File

The iTETRIS Applications Configuration File specifies the necessary values for the iCS to acquire knowledge concerning the cooperative traffic management applications that are going to be simulated in the iTETRIS platform. An example of the XML configuration file based on the ITS Cooperative Demo Application included with the iTETRIS source code is shown in the following:

```
<Applications>
	<Application>
		<name>DEMO APP</name>
		<executable>demoapp</executable>
		<ip>localhost</ip>
		<port>1986</port>
		<seed>123456</seed>
		<rate>0</rate>
		<result-container>OUTPUT_SET_SPEED_ADVICE_DEMO</result-container>
		<serviceId unicast="serviceIdUnicast"
			multicast="serviceIdMulticast"
			broadcast="serviceIdBroadcast"
			geobroadcast="serviceIdGeobroadcast"
			topobroadcast="serviceIdTopobroadcast"/>
		<stations>
		<id>5000</id>
		<id>6000</id>
		</stations>
	</Application>
	<!--
	<Application>
		Describe your other Application here
	</Application>
	-->
</Applications>
```

Similarly to the iTETRIS master and iCS configuration files, the iTETRIS Applications configuration file is also formatted using the XML language. It consists in one block (Table 7) that is repeated in the file as many times as the number of distinct applications that are going to be running in the iTETRIS platform.

TAG NAME	DESCRIPTION
Name	The name of Application as specified by the developer.
Executable	The command line string necessary to create the execution thread of the Application in the operating system.
Ip	The IP of the machine running the Application.
Port	Socket port number where the Application is expecting commands
Seed	Value to generate random numbers to decide if a vehicle will run the Application in the iTETRIS simulation.
Rate	Through this value, the iTETRIS user can define (out of the total number of vehicles) the percentage of vehicles that will run the Application in the iTETRIS simulation (Application penetration rate).
result-container	Specifies a set of values that correspond to the actions that the iCS has to undertake when receiving a result from the Application.

serviceId	In iTETRIS, the Application will use one or more of the transmission types or message type that can be simulated in ns-3. The traffic developer can reference the wanted transmission modes or message type by correctly setting these values according to the possibilities provided by ns-3. The parameters unicast, multicast, broadcast, geobroadcast, topobroadcast can be set to the ns-3 defined values serviceIdUnicast, serviceIdMulticast, serviceIdBroadcast, serviceIdGeobroadcast and serviceIdTopobroadcast, CAM, DENM.
Id	Although the iCS assigns Applications to vehicles in a random way, it is possible to manually define a set of stations that are forced to run the Application. This applies for example to the fixed stations defined in the Facilities configuration files (section 5.2.3). In this case, the id of these fixed stations is specified so that they will also run the cooperative Application. In this context, the id 0 is reserved as the id of the Traffic Management Center (TMC) node that can also run cooperative applications. The TMC is logically connected to all the fixed communications base stations deployed in the simulation scenario. However, the backbone wired connections of the TMC to and from these base stations are not modelled in iTETRIS. It is in fact supposed that TMC and base stations safely exchange messages.

Table 7. Applications configuration file tag summary.

5.4 ns-3 Configuration Files

ns-3 has to be properly configured in order to simulate any of the radio communications technologies implemented in the iTETRIS platform. To this aim, the ns-3 main configuration file is used. The ns-3 main configuration file is indicated in the iTETRIS Master Configuration File by the “communication-config-technologies-file” tag. Please note that in the current version of iTETRIS the name of ns-3 main configuration file MUST begin with the string “configTechnologies”.

An example of the content of the ns-3 main configuration file is presented in the following (configTechnologies-ics.xml):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ns3Configuration>
  <randomGenerators>
      <randomGenerator seed="1" runNumber="1" />
  </randomGenerators>
  <installers>
      <installer type="ns3::WaveVehicleInstaller" name="WaveVehicle"
file="confWaveVehicle.xml" default="false" />
      <installer type="ns3::WaveRsuInstaller" name="WaveRsu" file="confWaveRsu.xml"
relatedInstaller="WaveVehicle" default="false" />
      <installer type="ns3::UmtsBsInstaller" name="UmtsBs" file="confUmtsBs.xml"
default="false" />
      <installer type="ns3::UmtsVehicleInstaller" name="UmtsVehicle"
file="confUmtsVehicle.xml" default="false" />
      <installer type="ns3::WimaxBsInstaller" name="WimaxBs" file="confWimaxBs.xml"
default="false" />
      <installer type="ns3::WimaxVehicleInstaller" name="WimaxVehicle"
file="confWimaxVehicle.xml" default="false" />
      <installer type="ns3::DvbhVehicleInstaller" name="DvbhVehicle"
file="confDvbhVehicle.xml" default="false" />
      <installer type="ns3::DvbhBsInstaller" name="DvbhBs" file="confDvbhBs.xml"
```

```
default="false" />
      <installer type="ns3::TmcInstaller" name="TMC" default="false" />
      <installer type="ns3::ItetrisNetworkTransportInstaller" name="NetTrans"
default="true" />
      <installer type="ns3::MobilityModelInstaller" name="Mobility" default="true" />
  </installers>
</ns3Configuration>
```

This file is used to set the installers for any of the implemented radio communications technologies that want to be used in the simulation (corresponding to ns-3 NetDevices). Specific configuration files for each communications technology are indicated by the attribute “file”. These configuration files need to be located in the same folder as the ns-3 main configuration file. In the next subsections, a more detailed description of these configuration files is given[1].

5.4.1 ETSI ITS G5A Configuration Files

iTETRIS offers the possibility to install two ETSI ITS G5A (European version of the IEEE 802.11p standard) NetDevices on a given node, one for the control channel (CCH) and one able to switch over the service channels (SCHs). Moreover, every ITS G5A NetDevice has to be differently configured according with the type of node it is installed on, namely vehicles or Roadside Units (RSUs). “WaveVehicle” indicates an ITS G5A NetDevice installed over vehicles, while “WaveRsu” indicates an ITS G5A NetDevice installed over RSUs. Differently from the WaveVehicle NetDevice, the WaveRsu NetDevice is configured to present higher antennas heights. “confWaveVehicleWINNER.xml” and “confWaveRsu.xml” are two examples of configuration files for both types of Vehicle and RSU ITS G5A NetDevices. The next tables show the content of those files.

The following example corresponds to the WaveVehicle ITS G5A configuration file.

```
<waveVehicle>

  <yansWifiChannel>
    <propagationLoss name="ns3::WinnerB1LossModel"/>
    <propagationLoss attribute="Frequency" value="5.9e9"/>
    <propagationLoss attribute="EffEnvironmentHeight" value="1"/>
    <visibilityModel name="ns3::BuildingMapModel"/>
    <visibilityModel attribute="VisibilityMapFile" value="/home/user/iTETRIS/ns-
3/scratch/asymmetricbuildings.txt"/>
    <shadowingModel name="ns3::ShadowingModel"/>
    <shadowingModel attribute="CorrelatedShadowing" value="true"/>
    <fadingModel name="ns3::FadingModel"/>
    <fadingModel attribute="FadingFile" value="/home/user/iTETRIS/ns-
3/scratch//ricean_table.txt"/>
    <fadingModel attribute="MaxVelocity" value="50"/>
    <InterferenceRange attribute="Vehicle" value="500"/> <!-- set this value according
to the transmission power and the propagation model-->
    <InterferenceRange attribute="RSU" value="1200"/> <!-- set this value according to
the transmission power and the propagation model-->
  </yansWifiChannel>
  <yansWifiPhy>
    <wavePhyCCH attribute="TxGain" value="0.0"/>
    <wavePhyCCH attribute="RxGain" value="0.0"/>
    <wavePhyCCH attribute="TxPowerLevels" value="60"/>
    <wavePhyCCH attribute="TxPowerEnd" value="33"/>
    <wavePhyCCH attribute="TxPowerStart" value="20"/> <!-- Actual txon power (dBm)-->
    <wavePhySCH attribute="TxGain" value="0.0"/>
```

[1] To get familiar with ns-3 simulations and the use of the described communications technologies, iTETRIS user can find sample scripts to run simulations involving ns-3 only in the folder ns-3/scratch/

```
    <wavePhySCH attribute="RxGain" value="0.0"/>
    <wavePhySCH attribute="TxPowerLevels" value="60"/>
    <wavePhySCH attribute="TxPowerEnd" value="33"/>
    <wavePhySCH attribute="TxPowerStart" value="20"/>
  </yansWifiPhy>
  <qosWifiMac>
    <wifiRemoteStationManager attribute="RtsCtsThreshold" value="2000"/>
    <wifiRemoteStationManager attribute="FragmentationThreshold" value="2300"/>
  </qosWifiMac>
  <mobilityModel>
    <antenna attribute="AntennaHeight" value="1.5"/>
  </mobilityModel>
  <applications>
    <application>
      <C2CIP itetrisName="serviceIdUnicast"/>
      <C2CIP attribute="PortC2C" value="7090"/>
      <C2CIP attribute="PortIP" value="7091"/>
      <C2CIP attribute="Frequency" value="2"/>
      <C2CIP attribute="PacketSize" value="300"/>
    </application>
    <application>
      <C2CIP itetrisName="serviceIdTopoBroadcast"/>
      <C2CIP attribute="PortC2C" value="8083"/>
      <C2CIP attribute="PortIP" value="8082"/>
      <C2CIP attribute="Frequency" value="2"/>
      <C2CIP attribute="PacketSize" value="300"/>
    </application>
  </applications>
</waveVehicle>
```

As it can be observed, the configuration file is divided into several groups of parameters that are delimited by tags, “yansWifiChannel”, ”yansWifiPhy”, “qosWifiMac”, “mobilityModel” and “applications”. The group “yansWifiChannel” allows configuring the parameters related to the radio propagation channel. The main parameters in yansWifiChannel are:

1. propagationLoss: configures the propagation loss model to be used (in the example “WinnerB1”). The name of the propagation loss model and the set of attributes (e.g. Frequency in Hz, EffEnvironmentHeight in meters, etc.) have to be provided.
2. visibilityModel: configures the visibility model to be used. The visibility model (if supported by the propagationLoss model) is used to determine the LOS/NLOS conditions between transmitter and receiver, as they may considerably affect the radio propagation. In the example, the visibility model employed (BuildingMapModel) provides the locations of the buildings placed in the simulation scenario as specified by the “asymmetricbuildings.txt” (indicated by the complete path).
3. shadowingModel: configures the shadowing model to be used (if supported by the propagationLoss model). As shown in the example, in the case of the Winner model correlated and uncorrelated shadowing can be simulated.
4. fadingModel: configures the multipath fading model used by the propagationLoss model (if supported by it). As shown in the example, in the case of the Winner model, a file ricean_table.txt” (indicated using the complete path) has to be specified, along with the maximum velocity (in km/h) that is expected to be reached by vehicles in the simulation scenario (MaxVelocity). Both these parameters are used by the model to compute the fading variable for propagation loss computation.
5. InterferenceRange: configures the interference range for Vehicle-to-Vehicle (V2V) and Vehicle-to-RSU (V2I) transmissions. The interference range is the maximum distance separating transmitting and receiving nodes that the ns-3 simulator considers for packet reception computation. These values of interference range for both V2V

(attribute="Vehicle") and V2I (attribute="RSU") transmissions have to be set considering that, fixing a given propagationLoss model and a transmission power, the probability of packet receptions beyond a given distance is negligible (in the example 500 and 1200 meters can be used for the WinnerB1 model and a transmission power of 20 dBm). This solution permits to highly reduce the simulation time given that the computations to determine if a message is correctly received are made only for nodes within the interference range of the transmitter and not for all the nodes in the simulation scenario.

The group “yansWifiPhy” allows configuring the parameters related to the PHY layer of an ITS G5A NetDevice. Different parameters can be selected for the ITS G5A NetDevices operating on CCH and the SCH. The main parameters in “yansWifiPhy” are:

1. TxGain and RxGain (in dB).
2. TxPowerLevels (in integer values).
3. TxPowerEnd (in dBm): maximum transmission power.
4. TxPowerStart (in dBm): minimum transmission power. This transmission power is the one employed in the simulation. If the transmission power wants to be changed during the simulation, it has to be specified in a per-packet basis though the ns-3 packet tag mechanism (using the transmission-power packet tag).

The group qosWifiMac allows configuring the parameters related to the MAC layer, such as RtsCtsThreshold and FragmentationThreshold. The group mobilityModel allows configuring the antenna height (AntennaHeight in meters) of the vehicle. The set of parameters included in the group applications is related to the creation of ns-3 applications used by the simulator to transmit and receive packets over the nodes equipped with ITS G5A. The value of the parameter “itetrisName” must correspond to the “serviced” value specified by the iTETRIS Application Block. This is required for the iTETRIS Application Block to reference an ns-3 application installed over a given node and activate the corresponding type of transmission over it

The next table shows the configuration file of a WaveRsu ITS G5A NetDevice. As it can be seen, it is similar to the configuration file of the WaveVehicle ITS G5A NetDevice, in fact it has exactly the same tags. However, in this case there is no need to define the parameters to configure the radio propagation channel, since for WaveRsu NetDevices the same propagation channel as configured for the WaveVehicle will be installed, as specified in the ns-3 main configuration file in the instruction: installer type="ns3::WaveRsuInstaller" name="WaveRsu" file="confWaveRsu.xml" relatedInstaller="WaveVehicle". Please also note that in the mobilityModel information the antenna height for a WaveRsu NetDevice is set to a higher value compared to a WaveVehicle NetDevice.

```
<waveVehicle>
    <yansWifiPhy>
      <wavePhyCCH attribute="TxGain" value="0.0"/>
      <wavePhyCCH attribute="RxGain" value="0.0"/>
      <wavePhyCCH attribute="TxPowerLevels" value="60"/>
      <wavePhyCCH attribute="TxPowerEnd" value="33"/>
      <wavePhyCCH attribute="TxPowerStart" value="20"/> <!-- Actual txon power (dBm)-->
      <wavePhySCH attribute="TxGain" value="0.0"/>
      <wavePhySCH attribute="RxGain" value="0.0"/>
      <wavePhySCH attribute="TxPowerLevels" value="60"/>
      <wavePhySCH attribute="TxPowerEnd" value="33"/>
      <wavePhySCH attribute="TxPowerStart" value="20"/>
    </yansWifiPhy>
    <mobilityModel>
      <antenna attribute="AntennaHeight" value="6"/>
    </mobilityModel>
    <applications>
     <application>
      <C2CIP itetrisName="serviceIdUnicast"/>
      <C2CIP attribute="PortC2C" value="7090"/>
      <C2CIP attribute="PortIP" value="7091"/>
      <C2CIP attribute="Frequency" value="2"/>
      <C2CIP attribute="PacketSize" value="300"/>
     </application>
     <application>
      <C2CIP itetrisName="serviceIdTopobroadcast"/>
      <C2CIP attribute="PortC2C" value="8083"/>
      <C2CIP attribute="PortIP" value="8082"/>
      <C2CIP attribute="Frequency" value="2"/>
      <C2CIP attribute="PacketSize" value="300"/>
    </application>
  </applications>

</waveVehicle>
```

5.4.2 UMTS Configuration Files

In iTETRIS, two types of UMTS nodes can be specified: UMTS vehicles and fixed UMTS stations (UMTS Nodes B). This will result into two different ns-3 NetDevices modelling the Physical and Data Link layers of the UMTS technology installed on a node. In order to simplify the configuration of each type, two different configuration files need to be indicated in the ns-3 main configuration file ("confUmtsVehicle.xml" and "confUmtsBs.xml" in the previous example). The content of each file is represented in the following.

The first example shows the content of the "confUmtsVehicle.xml" file.

```
<umtsVehicle>
   <UmtsPhy name="NodeUE">
    <phy attribute="AntennaHeight" value="2.0"/>
    <phy attribute="AntennaGain" value="2.0"/>
    <phy attribute="SystemLoss" value="0"/>
    <phy attribute="DedicatedTxPower" value="-15"/>
    <phy attribute="TxFrequency" value="2000000000"/>
    <phy attribute="CommonTxPower" value="-21"/>
    <phy attribute="MinDistance" value="15"/>
    <phy attribute="TargetSNR" value="100"/>
    <phy attribute="PowerControlStep" value="step1"/>
  </UmtsPhy>
  <applications>
    <application>
      <UmtsApp itetrisName="Unicast"/>
      <UmtsApp attribute="PortIP" value="7080"/>
      <UmtsApp attribute="Frequency" value="2"/>
      <UmtsApp attribute="PacketSize" value="30"/>
      <UmtsApp attribute="ApplicationType" value="AM"/>
    </application>
  </applications>
</umtsVehicle>
```

The configurable parameters:

1. Antenna Height (in meters), and Antenna Gain (in dB).
2. System Loss (in dB).

3. Dedicated Transmission Power (the transmission power used for the DCH Channels) and Common Transmission Power (the transmission power used for the RACH/FACH Channels) (in dB).
4. Transmission Frequency (in Hz).
5. Minimum distance. The minimum distance between two nodes for which path losses can be assumed (in meters).
6. TargetSNR. It indicates the desired value of the SNR (Linear value).
7. PowerControlStep. It controls how the transmission power is increased/decreased. The possible values are "step1" for increasing/decreasing 1dB, "step2" for increasing/decreasing 2dBs and "step3" for increasing/decreasing 3dBs.

The next set of parameters is related to the creation of ns-3 applications used by the simulator to transmit and receive packets over the nodes equipped with the UMTS communications technology. Each of these ns-3 application is used to define a specific type of UMTS transmission. The value of the parameter "itetrisName" must correspond to the "serviced" value specified by the iTETRIS Application Block. This is required for the iTETRIS Application Block to reference an ns-3 application installed over a given node and activate the corresponding type of transmission over it. The "ApplicationType" option marks the type of transmission. The possible values are:

- AM. Acknowledged Mode. This application implements the ACK flow control and the Fragmentation functionality.
- UM-FRAG. Unacknowledged mode without fragmentation.
- UM-NON_FRAG. Unacknowledged mode without fragmentation.

The rest of attributes are "PortIP" indicating the port the ns-3 application is going to be connected to for listening (each ns-3 application has to use a different port), the frequency by which messages are going to be transmitted in case of periodic transmissions (in Hz; for the unicast mode the frequency is set to 0 by default, indicating the transmission of only one message) and the message size.

The next example shows the content of the "confUmtsBs.xml" configuration file. Unlike the UMTS vehicle configuration file, for a fixed UMTS node B, a "shadowing" value (in dB) employed in the calculation of the path loss has to be specified.

```
<umtsBs>
   <UmtsPhy name="NodeB">
    <phy attribute="AntennaHeight" value="20.0"/>
    <phy attribute="AntennaGain" value="10.0"/>
    <phy attribute="SystemLoss" value="0"/>
    <phy attribute="DedicatedTxPower" value="-15"/>
    <phy attribute="TxFrequency" value="2000000000"/>
    <phy attribute="CommonTxPower" value="-21"/>
    <phy attribute="MinDistance" value="15"/>
    <phy attribute="Shadowing" value="10"/>
    <phy attribute="TargetSNR" value="100"/>
    <phy attribute="PowerControlStep" value="step1"/>
  </UmtsPhy>
   <applications>
    <application>
      <UmtsApp itetrisName="Unicast"/>
      <UmtsApp attribute="PortIP" value="7080"/>
      <UmtsApp attribute="Frequency" value="2"/>
```

```
       <UmtsApp attribute="PacketSize" value="30"/>
       <UmtsApp attribute="ApplicationType" value="UM-NON_FRAG"/>
     </application>
   </applications>
</umtsBs>
```

5.4.3 DVB-H Configuration Files

DVB-H configuration files are also different according to the type of node they refer to: "confDvbhVehicle.xml" and "confDvbhBs.xml". The next examples show the content of those files.

The first one corresponds to values used to specify a DVB-H Base Station's properties:

```
<dvbhBs>
   <DvbhOfdm name="NodeB">
    <ofdm attribute="AntennaHeight" value="11.0"/>
    <ofdm attribute="AntennaGain" value="10.0"/>
    <ofdm attribute="TxPower" value="-15"/>
    <ofdm attribute="TxFrequency" value="1300.0"/>
    <ofdm attribute="TotalInterference" value="-80"/>
  </DvbhOfdm>
   <applications>
    <application>
      <DvbhApp itetrisName="Multicast"/>
      <DvbhApp attribute="PortIP" value="8080"/>
      <DvbhApp attribute="Frequency" value="2"/>
      <DvbhApp attribute="BurstSize" value="30000"/>
      <DvbhApp attribute="ApplicationType" value="BROADCAST"/>
      <DvbhApp attribute="ServiceIp" value="255.255.255.255"/>
      <DvbhApp attribute="ServiceId" value="10"/>
        <streams>
            <servicestream itetrisName="Audio" StreamId="1100"/>
            <servicestream itetrisName="Video" StreamId="1101"/>
        </streams>
    </application>
  </applications>
</dvbhBs>
```

The set of parameters correspond to the OFDM Layer and are listed below:

1. Antenna Height (in meters), and Antenna Gain (in dB).
2. Transmission Power (dB).
3. Transmission Frequency (MHz).
4. Total Interference. In order to simplify the operations in DVB-H the interference level is set to a static value introduced by this parameter.

As for UMTS, it is necessary to establish the list of ns-3 applications to install over the nodes that are going to be equipped with the DVB-H communications technology. There are two types of ns-3 applications (MULTICAST and BROADCAST); the "ServiceIP" and "ServiceID" values must be unique for each ns-3 application in order to distinguish them. Each application needs to include one or more streams defined within the tag "streams". One service is defined by its streams (for instance, the TV service could have the following streams: Audio, Video and Subtitles. The number of streams will impact in the number data messages to be transmitted; if the service needs to send the subtitles it will require the transmission of more data.). Each stream needs to be defined with a different "itetrisName" and "StreamId".

The configuration of the DVB-H communications technology over vehicles is similar to the configuration of fixed stations, but in this case there is no need to define the service streams:

```
<dvbhVehicle>
   <DvbhOfdm name="NodeUE">
    <ofdm attribute="AntennaHeight" value="2.0"/>
    <ofdm attribute="AntennaGain" value="2.0"/>
    <ofdm attribute="SystemLoss" value="0"/>
    <ofdm attribute="Shadowing" value="5"/>
    <ofdm attribute="TxPower" value="-15"/>
    <ofdm attribute="TxFrequency" value="1300"/>
    <ofdm attribute="MinDistance" value="1000"/>
    <ofdm attribute="TotalInterference" value="-70"/>
  </DvbhOfdm>
   <applications>
    <application>
      <DvbhApp itetrisName="Unicast"/>
      <DvbhApp attribute="PortIP" value="8080"/>
      <DvbhApp attribute="ApplicationType" value="BROADCAST"/>
      <DvbhApp attribute="ServiceIp" value="255.255.255.255"/>
      <DvbhApp attribute="ServiceId" value="10"/>
    </application>
  </applications>
</dvbhVehicle>
```

5.4.4 WiMAX Configuration Files

The configuration process of the WiMAX technology follows an approach similar to that presented for the UMTS and DVB-H technologies. Two different types of configuration files are used, one to configure WiMAX base stations and one to configure WiMAX vehicles. In the following examples, the content of the configuration files for both for both types of WiMAX nodes are provided.

The first one corresponds to the WiMAX Base Station's configuration.

```
<Wimax>
   <WimaxPhy>
    <phy attribute="Frequency" value="3500000000"/>
    <phy attribute="Bandwidth" value="20000000"/>
    <phy attribute="TxPower" value="30"/>
    <phy attribute="TxGain" value="0"/>
    <phy attribute="RxGain" value="0"/>
    <phy attribute="NoiseFigure" value="5"/>
    <phy attribute="CoverageRange" value="5000"/>
    <phy attribute="BSAntennaHeight" value="30"/>
    <phy attribute="SSAntennaHeight" value="1.5"/>
  </WimaxPhy>
  <ipConfiguration>
    <ip address="10.4.0.0" mask="255.255.0.0"/>
  </ipConfiguration>
  <applications>
   <application>
     <WimaxApp itetrisName="serviceIdUnicast"/>
     <WimaxApp attribute="PortIP" value="70"/>
     <WimaxApp attribute="Frequency" value="2"/>
     <WimaxApp attribute="PacketSize" value="30"/>
   </application>
  </applications>
</Wimax>
```

The set of parameters in the group “WimaxPhy” correspond to the WiMAX PHY layer and are listed below.

5. Transmission Frequency (in Hz)
6. Transmission Bandwidth (in Hz)
7. Transmission Power (in dB)
8. Tx and Rx Gain (in dB)
9. Noise Figure (in dB)
10. Coverage Range (in m)
11. BS and SS Antenna’s Height (m)

The set of configuration parameters tagged by “ipConfiguration” are related to the network layer. The IP base address and mask of the network are specified. As for the previous examples it is also necessary to provide the parameters to configure the ns-3 applications that WiMAX stations are going to use to transmit messages over the WiMAX communications technology. Among the set of parameters, the iTETRISName of the service, the PortIP, the message generation Frequency, and the PacketSize of messages should be specified.

The configuration of the WiMAX communications technology over vehicles is similar to the configuration of the WiMAX fixed stations, but in this case there is no need to provide the IP parameters:

```
<Wimax>
   <WimaxPhy>
    <phy attribute="Frequency" value="3500000000"/>
    <phy attribute="Bandwidth" value="20000000"/>
    <phy attribute="TxPower" value="30"/>
    <phy attribute="TxGain" value="0"/>
    <phy attribute="RxGain" value="0"/>
    <phy attribute="NoiseFigure" value="5"/>
    <phy attribute="CoverageRange" value="5000"/>
    <phy attribute="BSAntennaHeight" value="30"/>
    <phy attribute="SSAntennaHeight" value="1.5"/>
  </WimaxPhy>
  <applications>
   <application>
     <WimaxApp itetrisName="serviceIdUnicast"/>
     <WimaxApp attribute="PortIP" value="70"/>
     <WimaxApp attribute="Frequency" value="2"/>
     <WimaxApp attribute="PacketSize" value="30"/>
   </application>
  </applications>
</Wimax>
```

6 APPLICATION INSTALLATION INSTRUCTIONS

Applications can be installed and run in various languages- Java, Python or C++. They can be compiled and installed independently from the rest of the iTETRIS blocks, but you have to install the required compilers as shown in the following example:

```
sudo apt-get install build-essential python openjdk-6-jdk
```

7 iCS STRUCTURAL DESCRIPTION

The iCS source code is included in the iCS/src/ics folder of the iTETRIS distribution. It includes the source code of the iCS core functions responsible of configuring, synchronizing and running the simulation. Furthermore, it is divided in subfolders (Figure 3) containing different strategic components of the program such as:

- **applications_manager**: controls the iCS communications with the Applications by defining a subscriptions and result containers mechanism (see section 9).
- **traffic_sim_communicator**: defines the classes that handle the communications with SUMO through the TraCI (Traffic Control Interface).
- **wirelesscom_sim_communicator**: defines the classes that handle the communications with ns-3 via the iNCI (iTETRIS Network simulator Control Interface).
- **wirelesscom_sim_message_tracker**: defines the mechanisms managing the control of the messages that are scheduled for ns-3 simulation of transmission.
- **facilities**: defines the ETSI ITSC Facilities implemented in the iCS.

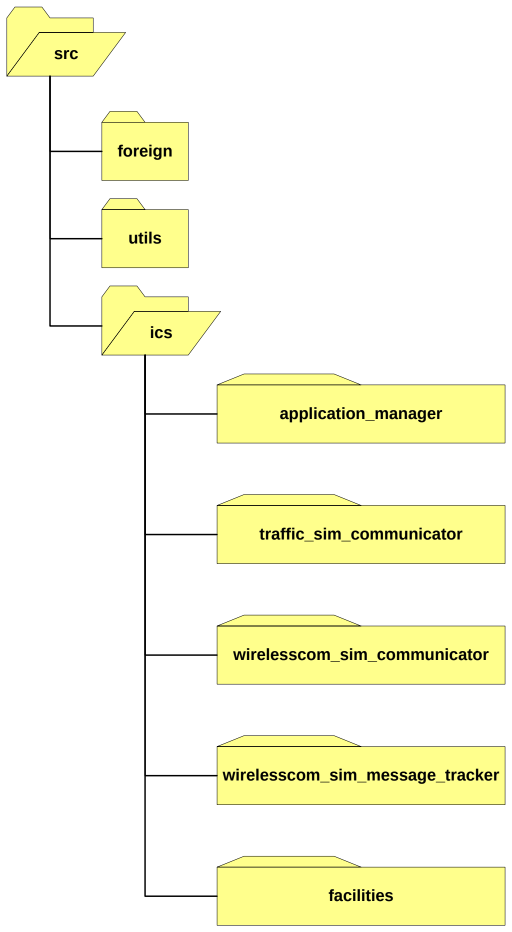


Figure 3. iCS source code directory schema.

8 BUILD AND INSTALL THE iTETRIS ITS COOPERATIVE DEMO APPLICATION

8.1 What is the iTETRIS ITS Cooperative Demo Application?

The ITS Cooperative Demo Application is a piece of software that illustrates how ITS systems developed by a third party can be integrated and evaluated in iTETRIS. Two different versions of Demo Applications are available in the current iTETRIS release. The source code along with the configuration files to run these applications in iTETRIS are located in the folders community-demo-app/ and community-demo-app-v2/ of the iTETRIS distribution.

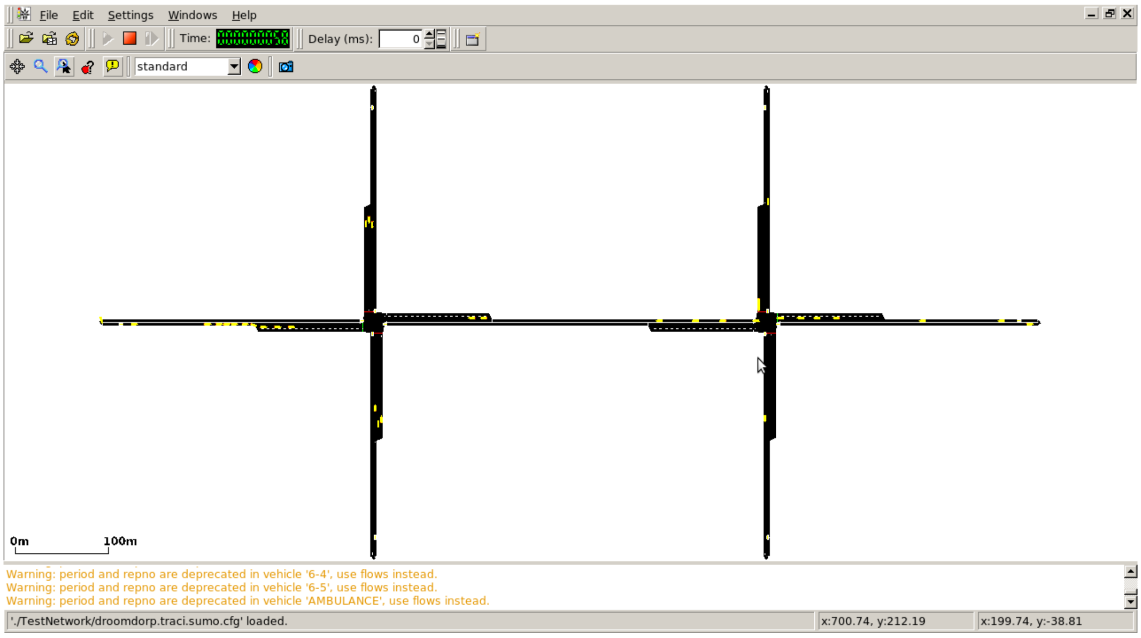
8.2 What is the impact of the iTETRIS ITS Cooperative Demo Application in the scenario?

The ITS Cooperative Demo Application executes over vehicles and over an RSU both equipped with ETSI ITS G5A technology. The user can also define the area over which vehicles transmit CAM messages[2]. The fixed RSU is able to transmit speed advices to vehicles that go through a user defined area (indicated by the parameters xcar, ycar and radiuscar as explained in section 8.7). Vehicles that receive the speed advice accordingly change their speed. In order to make the application impact more visible, the value contained in the speed advice is set to 0. In this way, vehicles receiving the speed advice will stop by setting their speed to 0.

8.3 What does the scenario look like?

The network is composed of two road crosses as in Figure 4. In the left cross there is a fixed station or Road Side Unit (RSU) equipped with ITS G5A technology communication capabilities. The mobile stations (vehicles) are also equipped with the same communication technology. As it can be seen in Figure 4, vehicles approaching to the left intersection, stop before reaching it as they receive the speed advice containing a 0 value.

[2] Letting vehicles transmit CAM messages only in the specific part of the simulated road network over which the execution of the Application wants to be tested is a design choice to reduce the simulation time and the use of computational resources, thereby enabling large-scale simulations.



```
Warning: period and repno are deprecated in vehicle '6-4', use flows instead.
Warning: period and repno are deprecated in vehicle '6-5', use flows instead.
Warning: period and repno are deprecated in vehicle 'AMBULANCE', use flows instead.
'./TestNetwork/droomdorp.traci.sumo.cfg' loaded.    x:700.74, y:212.19    x:199.74, y:-38.81
```

Figure 4. The SUMO traffic simulation scenario adopted in the Demo Application.

The necessary files configuring the applications and the scenario are in the example/ directory of the Application packages (community-demo-app/ and community-demo-app-v2/).

8.4 How can I build the iTETRIS ITS Cooperative Demo Application?

The following assumes that the rest of the components of the iTETRIS platform (ns-3, SUMO and the iCS) are configured and installed in your system. To configure, compile and install the Demo Application in one of its two versions, from the Demo Application's folder (either community-demo-app/ or community-demo-app-v2/) enter:

```
sudo  make -f Makefile.cvs

sudo ./configure --with-xerces-libraries=/usr/local/lib --with-xerces-
includes=/usr/local/include/xercesc

sudo make install
```

8.5 How can I run the simulation?

With all the components installed your machine (ns3, SUMO, iCS and ITS Cooperative Demo Application), and from the Demo Application's example/ folder (either community-demo-app/example/ or community-demo-app-v2/example/) type in terminal:

```
iCS -c itetris-config-file.xml
```

By default, the simulation is run with the sumo-gui user interface’s mode active, so that the user can visualize the impact of the Demo Application on the traffic scenario (vehicles stopping at the left intersection).

8.6 I found a bug! How can I report it?

Contribute with your suggestions and enhancements posting in the iTETRIS 10 10 10 Community [http://www.ict-itetris.eu/10-10-10-community/].

8.7 Can I change the parameters of the application scenarios?

Yes. The ITS Cooperative Demo Application is configured with the XML formatted file ("itscoopdemoapp-config-file.xml" in the examples directory) with the following parameters:

Parameter	Description
Start	Sets the time step in which the application starts sending back results to the platform.
Socket	Sets the port number in which the application establishes communication with the iTETRIS platform.
Logfile	Defines the path in which the application will dump logging messages.
Xcam	Defines the X axis value of the CAM transmission area for the RSU.
Ycam	Defines the Y axis value of the CAM transmission area for the RSU.
Radiuscam	Defines the radius of the CAM transmission area for the RSU.
Xcar	Defines the X axis value of the area from which the RSU wants to be aware of the vehicles within.
Ycar	Defines the Y axis value of the area from which the RSU wants to be aware of the vehicles within.
Radiuscar	Defines the radius value of the area from which the RSU wants to be aware of the vehicles within.

Table 8. ITS Cooperative Demo Application configuration file parameters.

8.8 Why are there two different versions of the iTETRIS demo-App?

Two ways for ITS applications to interact with the iCS in iTETRIS are provided, one where both applications would require subscriptions and result containers to control its logics (in community-demo-app/) and one related to an approach where the Application would control all the steps of its logics with subscriptions only (in community-demo-app-v2/). Further details can be found in section 9.

8.9 Can iTETRIS run without GUI support?

The iTETRIS platform standard output is the Unix terminal and does not require any GUI support. However to improve the clarity of the demo-app, we chose to leave the GUI on.

If you prefer not to use the GUI support, or if your station is not compatible with the GUI support provided by SUMO, you can remove it by replacing the <*traffic-executable*> tag in the itetris-config-file.xml as follows:

SUMO GUI-support enabled:

```
<traffic-executable value="sumo-gui -c"/>
```

SUMO GUI-support disabled:

```
<traffic-executable value="sumo -c"/>
```

8.10 Can I modify the code?

Yes. The ITS Cooperative Demo Application is based in a few files:

File	Description
its_demo_app_main.cpp	The starting point of the Application.
server.{h,cpp}	The object that handles the communication with the iCS, and hence with the rest of iTETRIS.
application-logic.{h, cpp}	The object that implements the ITS logic of the Application.
app-commands-subscriptions-constants.h	A set of command/ID pairs. Used to agree on the messages exchanged between iTETRIS and the ITS Application server. These values MUST NOT be changed.

Table 9. ITS Cooperative Demo Application essential source files.

The ITS Cooperative Demo Application also makes use of other pieces of GPL software to handle the command line options, file and sockets management. Those can be found in “utils” and “foreign” folders but are not essential part of the ITS Cooperative Demo Application.

8.11 Documentation

The source files mentioned above are commented and a Doxygen file is provided (“itscoopdemo.doxyfile”) to create documentation. The files are placed under the “docs” folder.

8.12 License

The ITS Cooperative Demo Application is licensed under GPLv3.

9 HOW TO BUILD YOUR OWN ITS COOPERATIVE APPLICATION

The objective of this section is to provide guidance to integrate Applications in the iTETRIS platform.

Essentially, the iCS provides data to the Applications and the Applications return a value of the result that would have eventually impact in the traffic or wireless environment.

In real life the Application is installed in each station, so there are many “instances” of the same Application running on top each station. In iTETRIS there is no such, there is only one instance of the Application running in the system and is its duty to internally run as several replicated Applications.

Data the iCS provides is either retrieved from the traffic simulator, SUMO for iTETRIS (e.g. position or speed of the vehicles) or from the wireless communication simulator (e.g. application’s data retrieved through wireless message receptions). The transmission of those application data is simulated in ns-3 and the result of it (simulated receptions) is used by the iCS that provides the information requested by the Application.

Besides requesting data from and returning a result to the iCS, the Applications can establish other actions such as setting specific CAM transmission area. These special features are established with the purpose of optimizing the performance of the platform or cover issues out of the scope of the simple information gathering and result returning.

The iCS objects used for the interaction between the iCS and Applications are organized in two categories:

- **Subscriptions**: Implementing the mechanism through which the Applications can retrieve data from the iCS or impose actions to be executed over the traffic and wireless simulators.
- **Result Containers**: Implementing the mechanisms aimed at storing and handling at iCS level the outputs of the Applications.

9.1 Subscriptions

The iCS offers information to the Applications through subscriptions. There are as many subscriptions as information that can be requested by an Application (e.g. vehicle’s data or reception of a particular wireless message). Moreover, the subscriptions can be used by the Applications to trigger actions (e.g. CAM transmission activation) over the integrated wireless or traffic simulators.

A subscription is linked to each station (“ITetrisNode” class in Figure 5) running an Application that requires that subscription. An iTETRIS node can have many subscriptions linked as those required by the various Applications tested in iTETRIS.

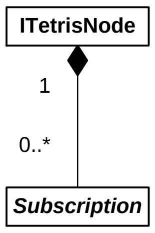


Figure 5. Relationship of the classes representing Subscription and the ITS station in the iCS.

In terms of coding, the different Subscription types inherit from the base class (“subscription.h”) to create subtypes of Subscription. See Figure 6.

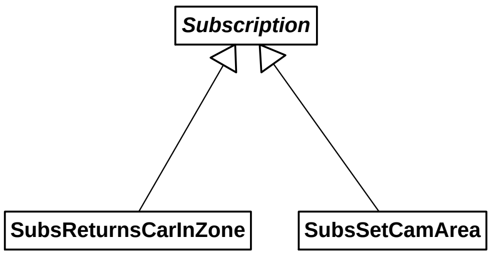


Figure 6. Generalization relationship between Subscriptions types and the base class.

9.2 Result Container

The idea behind of the Result Containers is to create an object to store at iCS level the outcome of an Application. The result can differ among Applications and therefore there are many Result Containers as the number of tested Applications. In some cases, the Result Container might directly apply the Application result's value, while in others cases it might require the simulation of a message transmission and a check to verify if that transmission successfully reached its destination. For example, in a real scenario the Application running in vehicle A might generate a result to be applied on vehicle B. For this reason, a message has to be transmitted with the result value from A to B and, if the transmission is successful, the vehicle B will make use of the result. Since in iTETRIS the Application is a unique block for every node, it has to be aware of such situations and therefore asks the iCS to trigger the simulation of the message transmission instead of directly applying the result in B.

A Result Containers is linked in the iCS to each iTETRIS node running an Application using that result container. An iTETRIS node can have many results containers linked as the number of Applications tested in iTETRIS. See Figure **¡Error! No se encuentra el origen de la referencia.**7.

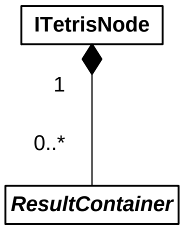


Figure 7. Relationship of the classes representing Result Container and the ITS station.

Finally each Result Container inherits from the Result Container base class ("result-container.h") to implement the appropriate actions according to a specific Application logic. See Figure **¡Error! No se encuentra el origen de la referencia.**8.

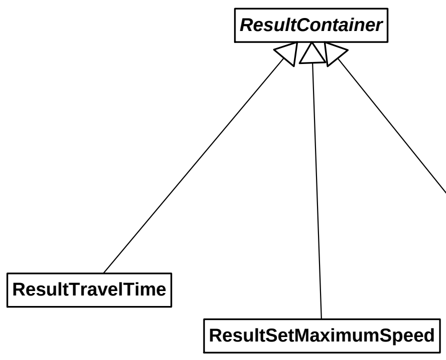


Figure 8. Generalization relationship between Result Container types and the base class.

9.3 Application Configuration File

The Application configuration file is the file used by the iCS to recognize the Application. The file is XML formatted and is explained in section 5.3.

9.4 iCS and Application Connection

To permit the connection between the iCS and the Application two objects are implemented in the iCS:

- **Application Handler** ("application-handler.h"): The Application Handler defines the time steps at which an Application shall be executed, and the data that has to be retrieved from the iCS for its execution. This information is defined based on the Application configuration files. To reduce the consumption of memory and computational resources, only one Application Handler is created for each Application. This approach also allows the iCS to distinguish between the many instances of the same application simultaneously running on different nodes. Thanks to the use of Application Handlers, the iCS just acts as an information supplier to the Application, and thereby does not need to be redefined for every new Application tested under iTETRIS. An Application Handler is linked in the iCS to each iTETRIS node running an Application using that Application Handler. An iTETRIS node can have many Application Handlers linked as the number of Applications tested in iTETRIS. See Figure **¡Error! No se encuentra el origen de la referencia.**9.

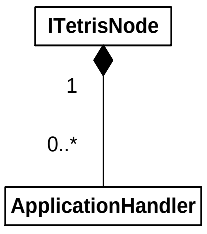


Figure 9. The ITS Stations have an Application Handler attached if they are running Applications.

- **Application Message Manager** or **Application Messenger** ("app-message-manager.h"). An Application Handler is connected to specific Application Messengers, which act as interfaces between the iCS and the Application. The Application Messenger is used to pack the information to be exchanged between the iCS and the

Application over an IP socket. This is in turn controlled by the Application Handler which implies there is one and only one socket per Application. See Figure 10**¡Error! No se encuentra el origen de la referencia.**.

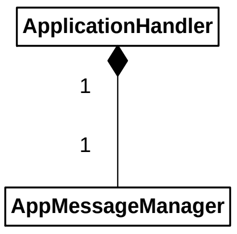


Figure 10. The Application Handler makes use of the "AppMessageManager" class to connect with the Application.

The primitives implemented by the Application Messenger are implemented in the "application-message-manager.cpp" file.

The iCS uses a Socket library to create the content of the primitives and transmit the data to the Application. The library is the same used by ns-3 and SUMO. If the Application is going to be written in "C++" we recommend using the very same library.

The library has two main objects:

- **The Socket**. This is a wrapper of the BSD Unix socket OS level functions.
- **Storage**. Under this name the native values such as Integer, Float, String, Bytes etc are packaged as single unit that is transmitted using the Socket.

9.4.1 Data Types

The length of the data types defined in the library:

DATA TYPE	SIZE	DESCRIPTION
Unsigned byte	8 bit	integer numbers (0 to 255)
byte	8 bit	integer numbers (-128 to 127)
integer	32 bit	integer numbers (-231 to 231-1)
float	32 bit	IEEE754 floating point numbers
double	64 bit	IEEE754 floating point numbers
string	variable	32 bit string length, followed by text coded as 8 bit ASCII
stringList	variable	32 bit string count n, followed by n strings

Table 10. Socket library recognized data types.

9.4.2 How to Calculate Command Length

A key part of the interaction between the iCS and the Application is the exchange of packets over Sockets. The socket sends a Storage object, which MUST contain a field called "Command Length" used by both sends (iCS and Application) to indicate how many bytes are expected to be sent. We describe next how this field is calculated..

Usually the parameter that contains this information is unsigned byte data type. In special cases, when the data length may exceed the maximum size of an unsigned byte, we may replace it by an unsigned integer. To do so, we still send an unsigned byte representing '0', and then an unsigned integer with the real length. The guidelines to get the final number is to count every bytes of the parameters including the one that contains the size. For instance, if a primitive has to send this information:

PARAMETER TYPE	CHARACTERS	SIZE	SIZE IN BYTES
Byte	-	8 bit	8/8 = 1 byte
Integer	-	32 bit	32/8 = 4 byte
Double	-	64 bit	64/8 = 8 byte
String	10	32 bit + (10 characters x 8 bit/character)	112/8 = 14 byte
String List	3 strings of characters: 5 10 15	32 bit + 32 bit + (5 characters x 8 bit/character) + 32 bit + (10 characters x 8 bit/character) + 32 bit + (15 characters x 8 bit/character)	32 + 72 + 112 + 152 = 368 368/8= 46 byte
TOTAL			73 byte

Table 11. Example of how to calculate a size of a command.

The amount calculated in the Table 11 refers to the size of the information the primitive tries to send to the other block. To this value, the parameter in which the size is actually stored has to be added (1 byte in case an unsigned byte data type is used, or 4 bytes in case an unsigned integer type is used).

9.5 Interactions between the iCS and Applications

The Applications Block of iTETRIS contains application logics specific to one or more ITS application. It uses data from iTETRIS and returns results to iTETRIS. In principle, one application has its own Application instance, and the interaction between different Applications can be handled through subscriptions. The next paragraphs describe how the iCS interacts with the Applications block in simulation time.

As illustrated in Figure 11, iTETRIS simulations consist of subsequent iterations of a loop in which ns-3, SUMO and the Applications are sequentially triggered by the iCS to execute their tasks. The total simulated time specified in the master configuration file is divided into simulation time steps of one second. For each simulation time step, the different iTETRIS blocks simulate all the application, traffic or wireless communications events scheduled for the corresponding time step. Thus, the iCS interacts with the Applications block at each time step. It loops all stations to query if the application logic of a particular Application has to take any action on it.

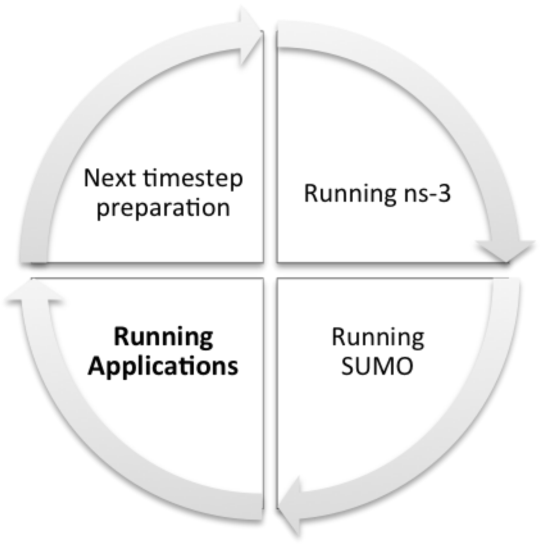


Figure 11. iTETRIS simulation loop.

This interaction involves the Subscriptions and Result Containers and is divided in five sub-steps as indicated in Figure 12:

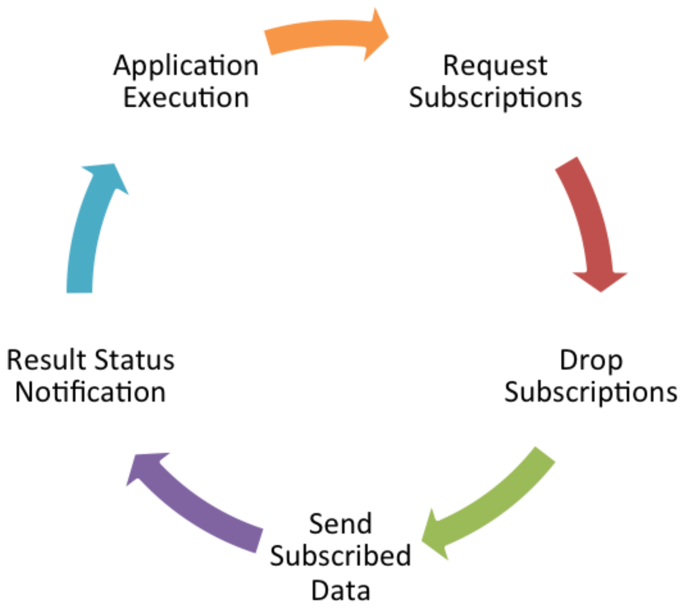


Figure 12. iTETRIS Running Application sub-cycle

9.5.1 Request Subscription

In this step the iCS queries all the iTETRIS nodes to check whether the Application installed over them needs to set a subscription. To do that, the Application Handler of the considered Application is used. The format of the primitive used for this request is as below.

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command (0x0A).
Current Time Step	Integer	The time step the simulation is sending the command.
Node ID	Integer	The iCS identifier associated to the node which is sending the command.

Table 12. Format of the primitive from iCS to Application to Request Subscription.

The expected response from the Application to the iCS has to match the following format. The response is divided in two primitives. The first one is related with the status of the response:

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command. The same value of the request, otherwise the iCS will report an error.
Result Type	Byte	Identifier of the response. Tells the iCS about the status of the actions taken in the Application side. There are three possible values: 0xFF: Error occurred in the Application side. 0x01: The operation was successful. 0x02: The command ID sent is not implemented.
Message	String	A description of the actions taken by the Application.

Table 13. Format of the response status of the primitive from Application to iCS for the Subscription request.

The second part of the response consists on a primitive that customizes the Subscription to be set in the iCS. For instance, the Application may request to set a subscription to activate CAM transmissions over a CAM Area as specified in the Application configuration file (see section 8.7).

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command. In this case 0x0A. (Or 0x0B to finish this step)
Subscription ID	Byte	The value that identifies the subscription univocally.
Parameters needed by the iCS to set the subscription (e.g. CAM Area parameters).		

Table 14. Format of the response data of the primitive from Application to iCS for the Subscription request.

It is important to highlight that the iCS will keep questioning the Application if it wants to add more subscriptions until the Application tells to stop. This is done by sending in the Command Id parameter of the previous table with the 0x0B value.

Please bear in mind that you should tell the iCS so if and only if you are sure that your application will never need another subscription during the application's runtime.

9.5.2 Drop Subscription

In this step the Application Handler loops over all the iTETRIS nodes, and for every subscriptions of a node, it queries the Application to know if the Subscription has to be deleted or will continue working for the next time step.

The "Subscription ID" is the same approach to identify the Subscription types used in the Request Subscription section. The format of the primitive is described in Table 15:

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command. The final value depends on the ID assigned to the subscription by the iCS developers.
Current Time Step	Integer	The time step the simulation is sending the command.
Node ID	Integer	The iCS identifier associated to the node which is sending the command.
Subscription ID	Byte	The value that identifies the Subscription type to drop univocally.

Table 15. Format of the primitive from iCS to Application to ask for a Drop Subscription.

The response is divided in two primitives. The first one is related with the status of the response:

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command. The same value of the request, otherwise the iCS will report an error.
Result Type	Byte	Identifier of the response. Tells the iCS about the status of the actions taken in the Application side. There are three possible values: 0xFF: Error occurred in the Application side. 0x01: The operation was successful. 0x02: The command ID sent is not implemented.
Message	String	A description of the actions taken by the Application.

Table 16. Format of the response status of the primitive from Application to iCS to Drop Subscription

The second part of the response consists in a primitive that gives the answer to the iCS:

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command. The same value of the request, otherwise the iCS will report an error.
Command Answer	Byte	Identifier of the response. Tells the iCS about the choice of the application related to the subscription:

		status of the actions taken in the Application side. There are three possible values: 0x0C: DROP_SUBSCRIPTION. 0x0D: RENEW_SUBSCRIPTION.

Table 17. Format of the response value of the primitive from Application to iCS to Drop Subscription.

9.5.3 Send Subscribed Data

In this step, the Application Handler loops over all the iTETRIS nodes and for all the subscriptions that are active on a node, detects the subscribed data to be transmitted from the iCS to the Application. To accomplish this task, there is one primitive per type of subscription implemented.

The format of the primitive depends on the Subscription. A specific "Command ID" is used to identify each primitive.

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command. The final value depends on the ID assigned to the subscription by the iCS developers.
Current Time Step	Integer	The time step the simulation is sending the command.
Node ID	Integer	The iCS identifier associated to the node which is sending the command.
Depends on Subscription		

Table 18. Format of the primitive from iCS to Application to send subscribed data.

The expected response format consists of one part concerning the response of the actions taken in the Application side.

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command. The same value of the request, otherwise the iCS will report an error.
Result Type	Byte	Identifier of the response. Tells the iCS about the status of the actions taken in the Application side. There are three possible values: 0xFF: Error occurred in the Application side. 0x01: The operation was successful. 0x02: The command ID sent is not implemented.
Message	String	A description of the actions taken by the Application.

Table 19. Format of the response status of the primitive from Application to iCS for subscribed data delivery.

9.5.4 Result Status Notification

In the previous paragraphs, it has been explained that some of the results returned by the Applications are not directly applicable and need to be handled by the result container (e.g. message to be simulated in the wireless simulator like for example a speed advice sent from an RSU to a vehicle). In this step the iCS informs the Application of the messages that were received correctly by their recipient nodes over the wireless simulator.

It is important to note that when the Application implies results that need to simulate specific message transmissions, then the Application has to assign an identifier to these transmissions. This identifier is needed by the iCS to tell the Application later on about the status of the transmission. The possible statuses are described in the following table and defined as "enum" in the "iCStypes.h" file:

NAME	VALUE	DESCRIPTION
kToBeScheduled	0	The Application requests the iCS to schedule the result for transmission in ns-3.
kScheduled	1	The result was scheduled in ns-3 and it is about to be transmitted or the transmission is in progress.
kArrived	2	The transmission was successful and the information reached the destination
kToBeApplied	3	The Application requests to the iCS that the value attached to the messages should be applied in the corresponding environment.
kToBeDiscarded	4	The Application requests to the iCS to discard the result and not to make use of the value.
kMissed	5	The transmission was unsuccessful and the information did not reach its destination. (Note: This status is not currently supported by the iCS)

Table 20. Summary of the possible statuses of a result.

The format of the primitive is as follows:

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command (0x14).
Current Time Step	Integer	The time step the simulation is sending the command.
Node ID	Integer	The iCS identifier associated to the node which is sending the command.
Number of Messages	Integer	The amount of messages to notify
Message ID	Integer	The ID of the message that was assigned by the Application.
Receiver ID	Integer	The ID of the station that received the Message ID associated to the previous parameter.

Table 21. Format of the primitive from iCS to Application to notify successful result transmissions.

The last two parameters are repeated as many times as defined by the parameter “Number of Messages”.

The response from the Application has a single primitive regarding the status of the actions taken in the Application with the following format:

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command. The same value of the request, otherwise the iCS will report an error.
Result Type	Byte	Identifier of the response. Tells the iCS about the status of the actions taken in the Application side. There are three possible values: 0xFF: Error occurred in the Application side. 0x01: The operation was successful. 0x02: The command ID sent is not implemented.
Message	String	A description of the actions taken by the Application.

Table 22. Format of the response status of the primitive from Application to iCS for result transmission notification.

Please note that these steps are never used if the second scheme of interaction between the Applications and iTETRIS as proposed in the community-demo-app-v2. In this second approach, only Application-based subscriptions are used. In this case, all data is handled from and by the applications, and as such any data brought back from iTETRIS to the Application are by subscriptions. As such, the results statuses are never transmitted to the result-container but stay in the application as the applications’ general state machine.

9.5.5 Application Execution

The last step is for the Application Handler to loops over all the iTETRIS nodes to tell the Application to execute its logics and to send back its result to the result container for storage. The format used for this interaction is as below (Table 23).

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command (0x14).
Current Time Step	Integer	The time step the simulation is sending the command.
Node ID	Integer	The iCS identifier associated to the node which is sending the command.

Table 23. Format of the primitive from iCS to Application to let the Application return result.

The response is divided in two parts. The firs primitive informs the iCS about the status of the actions taken in the Application:

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.

Command Id	Byte	The identifier of the command. The same value of the request, otherwise the iCS will report an error.
Result Type	Byte	Identifier of the response. Tells the iCS about the status of the actions taken in the Application side. There are three possible values: 0xFF: Error occurred in the Application side. 0x01: The operation was successful. 0x02: The command ID sent is not implemented.
Message	String	A description of the actions taken by the Application.

Table 24. Format of the response status of the primitive from Application to iCS for returning result.

The second part of the response is where the values of the result are going to be transmitted. The final format depends on the amount and information sent by the Application. Depending on the result, the values are processed by the function "ProcessResult()" that has to be implemented by the classes that inherit from the "ResultContainer" class ("app-result-container.cpp") .

Please note that in case the second approach for the interaction between the iCS and the Application is used (as in the community-demo-app-v2), the result container may still store the returned values, as for any other result container. The major difference is that it will store it in a specific table that can be used by the cross-application subscription (SUBS_X_APP_DATA) for another application to pull data from this result container and reuse it for its internal calculations.

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command. The same value of the request, otherwise the iCS will report an error.
Result Presence	Byte	The byte indicates if the Application has generated a result. There is result information: APP_RESULT_ON (0x0F) There is NO result information: APP_RESULT_OFF (0x10)
Values of the result of the Application.		

Table 25. Format of the data of the primitive from Application to iCS to process the result.

It is important to note that if the third parameter (Result Presence) indicates that there is no result for this command type, then the iCS will jump to the next command. If results are present, then a specific format for their processing at iCS level is used. As an example, the community-demo-app (returning speed advices to vehicles as results) uses a primitive that has the following complete format when it returns a result:

PARAMETER	TYPE	DESCRIPTION
Command Length	Integer	The size of the command in bytes.
Command Id	Byte	The identifier of the command. The same value of the request, otherwise the iCS will report an error.

Result Presence	Byte	The byte indicates if the Application has generated a result. There is result information: 0x0F There is NOT result information: 0x10
Number Of Results	Integer	The whole amount of "speed advice" values that have to be scheduled. Therefore the next parameters are repeated in the following order as many times as this parameter indicates.
Node ID	Integer	The ID of the node to which the Speed Advice has to be sent.
Maximum Speed	Float	The Speed Advice value.
Message ID	Integer	Because the Speed Advice is adressed to a station that will make use of the value, the iCS has to schedule a unicast message from the station that generated the Speed Advice to its destination. This ID identifies that unicast message.
Status	Integer	The meaning of this parameter was explained in the Table 20 of the 9.5.4 section. Basically it tells the iCS it has to process the result (schedule, directly apply, discard).
Frequency	Integer	These parameters are related to the wireless simulation of the unicast message transmission to simulate. The Application can set the frequency, calculate the payload length and establish the message regeneration time of the unicast message used to transmit the result.
Payload Length	Integer	
Message Regeneration Time	Float	

Table 26. Example of returning result from Application to iCS base on Speed Advice e Application.

A summary of the interactions between the iCS and the Application is illustrated in the following Figure 13:

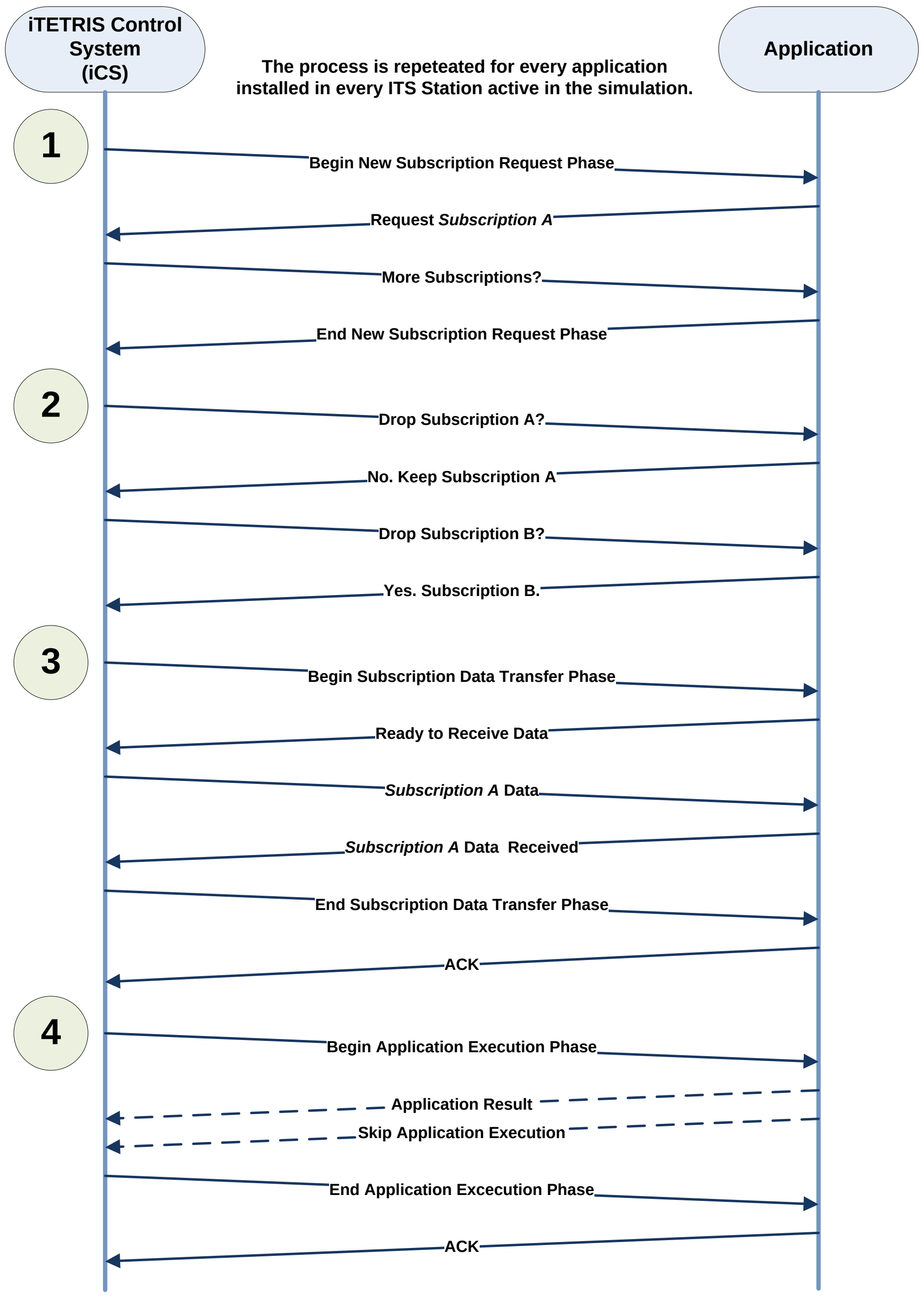

iTETRIS Control System (iCS)
Application
The process is repeteated for every application installed in every ITS Station active in the simulation.
1
Begin New Subscription Request Phase
Request Subscription A
More Subscriptions?
End New Subscription Request Phase
2
Drop Subscription A?
No. Keep Subscription A
Drop Subscription B?
Yes. Subscription B.
3
Begin Subscription Data Transfer Phase
Ready to Receive Data
Subscription A Data
Subscription A Data Received
End Subscription Data Transfer Phase
ACK
4
Begin Application Execution Phase
Application Result
Skip Application Execution
End Application Excecution Phase
ACK

Figure 13. Summary of the iCS and Application interaction.

9.6 Note on the Subscriptions and Result Containers

As described before, the interactions between iTETRIS and ITS applications are based on subscriptions and result containers primitives. Result containers play a potentially larger role than their name might indicate. Indeed, provided by the application designer, but located in the iCS, result containers may have access to deep iCS application logics, and as such, do not only store data, but may also act on behalf of the application in the iCS. The major limitation to this is that the application designer must be comfortable with deep iCS primitives.

Similarly, there might be one subscription representing any different specific data, but as such a new subscription should be integrated into the iCS for each new required data or action, which in turn also requires a deep understanding of the iCS deep primitives.

To ease the extensibility of the iTETRIS platform,, an alternative solution for ITS applications to interact with iTETRIS is provided.

A single generic result container capable of storing and retrieving variable data types is defined to have no interactions with the iCS so that the designer does not need to extend it (see Figure 14).

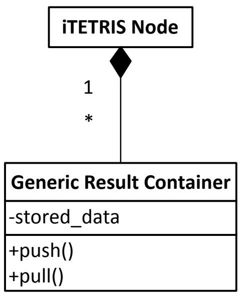


Figure 14. Generic Result Container used by all applications to store and retrieve generic type of data.

All interactions between the iCS and the Applications are instead managed through subscriptions. For this purpose, generic types of subscriptions are implemented, one for each interaction *type of interaction* (rather than type of data), and whose exact role is defined by the subscription's generic parameters. As such, the subscription does not represent a specific application or a data type, but rather an interaction type among different iTETRIS blocks as illustrated in Figure 15. These generic subscriptions are called Application-based Subscriptions, while the others are called iCS-based subscriptions.

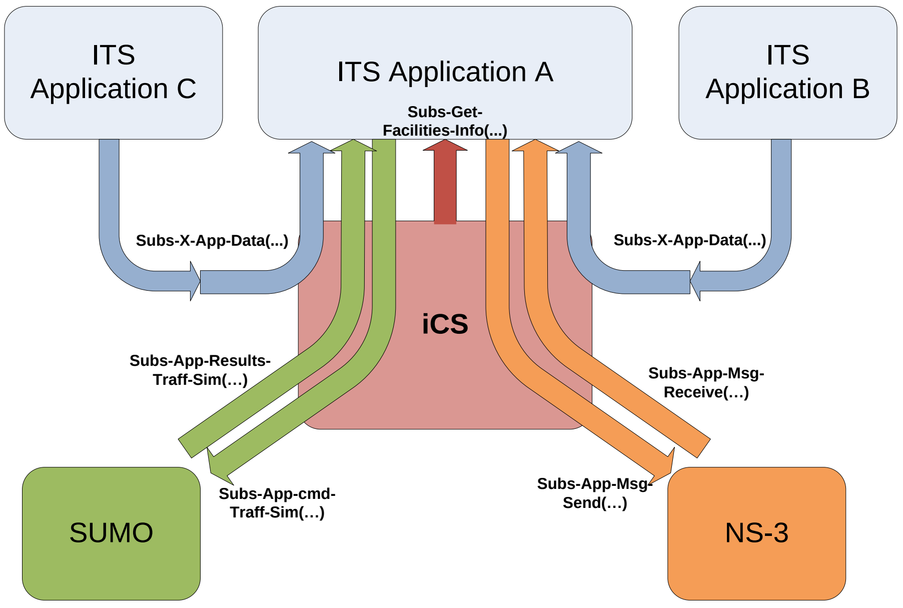


Figure 15. Illustration of the Six Application-based Subscriptions

The iTETRIS platform provides support for both interaction approaches. As an example, a specific Demo Application (described in section 8) has been implemented using both these two interaction approaches. These two different implementations are available in the community-demo-app (using the iCS-based subscriptions) and the community-demo-app-v2 (using the Application-based subscriptions).

In the community-demo-app, the interaction cycle between the Application and the iCS uses the same 5 steps as described in Figure 12. The Application uses the SUBS_CAM_AREA and RETURN_CAR_IN_ZONE subscriptions to activate transmission of CAM messages only in a specific area and for being notified about vehicles present in it. Transmissions of speed advices are triggered in the application logic only once for any vehicle present in the car area. The activation of the advised speed over the vehicles in the traffic simulator is then triggered in the iCS and more specifically in the Result Container of the Application, after checking the result status as described in Table 20.

In the community-demo-app-v2, the interaction cycle between the Application and the iCS uses the same 5 steps as described in Figure 12, but due to the specific usage of subscriptions to act on the iTETRIS blocks, actions are *'scheduled'* but only executed in the first step by sending the required subscriptions.

- **Request Subscriptions** – used to:
  - Set a recurring (at each simulation time step) SUBS_APP_MSG_RECEIVE subscription to notify the Application of the transmitting node (in this case the RSU) about correct receptions of speed advice messages at recipient nodes (in this case vehicles)
  - Set SUBS_CAM_AREA and RETURN_CAR_IN_ZONE subscriptions used to activate transmission of CAMs only in a specific area and notify the transmitting node (in this case the RSU) about vehicles present in it.
  - Set SUBS_APP_MSG_SEND subscription to let the Application of the transmitting node (in this case the RSU) trigger the simulation of transmission of speed advice messages to vehicles located over a specific area
  - Set a SUBS_CMD_TRAFF_SIM subscription to change the speed of vehicles when they receive a speed advice message
- **Drop Subscriptions** – used to:
  - Schedule a drop of the SUBS_APP_MSG_SEND and SUBS_CMD_TRAFF_SIM as they are one shot subscriptions
- **Send Subscribed Data**: used to:
  - Send all data gathered by the previously scheduled subscriptions, in particular: transfer received speed advices messages to the Application.
- **Result Status Notification**: never used
- **Application Execution** – used to:
  - Execute the application logic and 'schedule' the actions on the iTETRIS blocks (which will be executed in the first step)
  - Only optionally, return data to the GENERIC_RESULT_CONTAINER for storage

The order of scheduling and dropping subscriptions is given by the same finite state machine as described by application results as described in Table 20.

9.7 Constants

In order to distinguish each primitive, a series of identifiers are defined. The constants are stored in the iCS in the "app-commands-subscriptions-constants.h" header file. These constants MUST match that specified in the "app-commands-subscriptions-constants.h" on the application side.

9.8 Subscription Catalogue

The iCS version 0.3.0 is delivered with the following subscriptions. You may use any of them with your own ITS Cooperative Application. For the sake of clarity, we separated them in two classes, either iCS-based, or Application-based.

ICS-BASED SUBSCRIPTIONS		
VALUE	**DEFINE NAME (C++)**	**DESCRIPTION**
0x02	SUB_RETURNS_CARS_IN_ZONE	This Subscription returns the mobile stations that are currently present over a configurable area according to the information stored in the iCS Facilities
0x11	SUB_SET_CAM_AREA	The Applications can set a CAM transmission area with this Subscription. The area defines potential reception coverage of CAM messages from a particular station (the subscriber) by the stations inside of the area. They are used by the iCS to minimize the simulation time and the computational resources needed by ns-3. This Subscription does not return any kind of data.
0x12	SUB_TRAVEL_TIME_ESTIMATION_START	With this Subscription the Application sets the subscriber as geobroadcasting station over a given geobroadcasting area. The Receivers of the geobroadcast message will be forwarded to the Application.
0x13	SUB_TRAVEL_TIME_ESTIMATION_END	This Subscription works with the SUB_TRAVEL_TIME_ESTIMATION_ START. In the same way, the Application sets a geobroadcasting area and the receivers of the message will be

		captured by the iCS and then notified to the Application.
0x19	SUB_RECEIVED_CAM_INFO	This subscription is set to retrieve the data contained in received CAM messages.

APPLICATION-BASED SUBSCRIPTIONS		
VALUE	**DEFINE NAME (C++)**	**DESCRIPTION**
0x1C	SUBS_GET_FACILITIES_INFO	This Subscription returns requested data available at the facilities layer. Currently supported data types are: Awareness data from either CAM or from topological map.
0x2A	SUBS_APP_CMD-TRAFF-SIM	The Applications can command the traffic simulator to conduct a particular action as specified in the subscription's parameter field. Currently supported commands are *re-routing*, *changing speed*, *assigning road weights*
0x2C	SUBS_APP_RESULT-TRAFF-SIM	The Applications can request data from the traffic simulator for its internal logic as specified in the subscription's parameter field. Currently supported data are *current route*, *road weights*.
0x1E	SUBS_APP_MSG_SEND	This is a generic subscription for Applications to command the communication simulator to send messages. The type and configuration of the message is specified in the subscription's parameter field. Currently supported transmissions are UNICAST, GEO_UNICAST, GEO_BROACAST and TOPO_BROADCAST.
0x3A	SUBS_APP_MSG_RECEIVE	This is a generic subscription for applications to be notified about received messages. Depending on the

		subscription's parameter field, it can select to only receive message from or to a particular source/destination or be ubiquitous. Note that for receiving messages transmitted by the SUBS_APP_MSG_SEND subscription, a SUBS_APP_MSG_ RECEIVE MUST be scheduled.
0x2E	SUBS_X_APP_DATA	This subscription allows the exchange of data between different applications using the target application's result container. The type of data is specified in the subscription's parameter field.