Creating a Mission
Step by Step

Preface

Hello,
Creating a simple or complex mission, needs some thought and basic organisation before “diving in” with the actual work. If you have created a mission with the previous plugin, Mission-X v2.x, then you should have a good grasp of the basic concepts.
In Mission-X v3 there are some fundamental changes to the layout of the mission file, and the way we implement conditions and execute actions (like messages).

Enjoy
Saar Nagar - Snagar

Requisites

Mission-X v3 includes embedded scripting capabilities, this means that complex logic and most actions can be handled in a code, but there is enough features supported in the mission XML “language”, therefore I suggest to first try and build simple missions without scripts and only when you feel comfortable start to add scripts as complementary to complex logic.

The plugin uses MY-BASIC embedded scripting as the “scripting language” of choice, I think that the language is:

- Easy to grasp.
- Has simple syntax.
- Does a good enough job in terms of performance, even though it is an interpreter.

But, since it is a scripting language there are some rules, well… basic rules (no pun intended) how the plugin expects us to use it.
If you feel curious enough then head to the “designer guide” and search for MY-BASIC topic.

Other than the ability to use scripts, the rest of mission creation is laid out in an XML file which means it should be readable without special interpretations (see designer guide).

Tip:
In some elements there will be attributes you will not use. You can ignore them if they are optional (you do not have to write them down too).

Since v3.0.242.2 mission file version must be: “301”

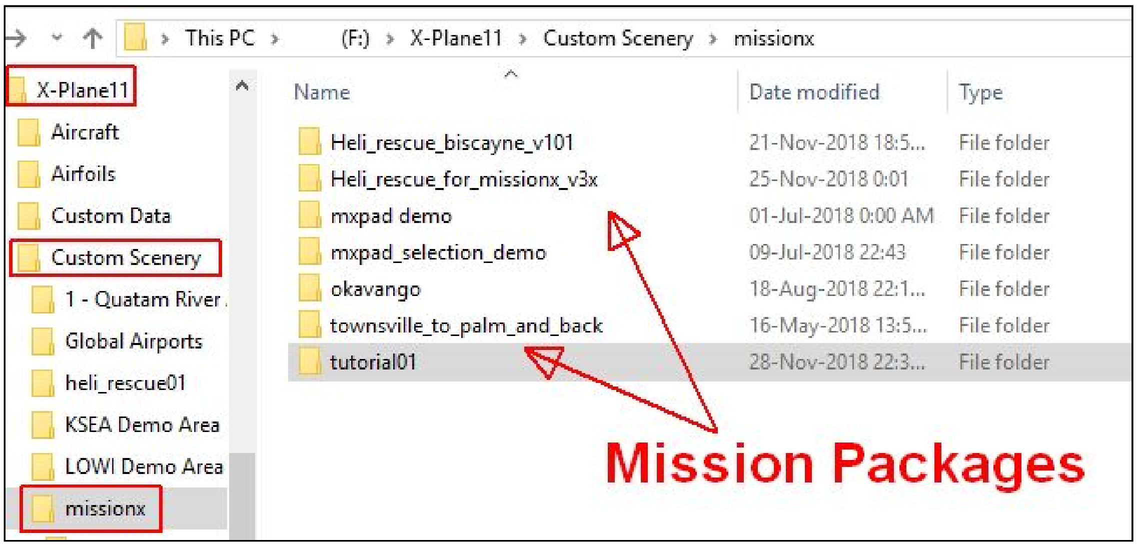
The Three Main Components to Build a Mission

Every mission should have the following:

1. Mission file.
2. Mission Folder Tree (packaging).
3. Mission specific resources (custom scenery, scripts, images, sound files…).

It might sound trivial, yet it is important to understand that there is also a shift in the way we package our mission. Everything, except savepoints, should be located in one place, the mission folder.
Here are the mandatory requirements for every mission (as of v3.0.161)

1. The root of any mission package (as of v3.0.160) is:
   "{X-Plane folder} /Custom Scenery/missionx" folder



Name	Date modified	Type
Heli_rescue_biscayne_v101	21-Nov-2018 18:5...	File folder
Heli_rescue_for_missionx_v3x	25-Nov-2018 0:01	File folder
mxpad demo	01-Jul-2018 0:00 AM	File folder
mxpad_selection_demo	09-Jul-2018 22:43	File folder
okavango	18-Aug-2018 22:1...	File folder
townsville_to_palm_and_back	16-May-2018 13:5...	File folder
tutorial01	28-Nov-2018 22:3...	File folder

2. You can package few mission files in the same mission folder. This will allow you to share resources, and make different level of difficulties of same mission.

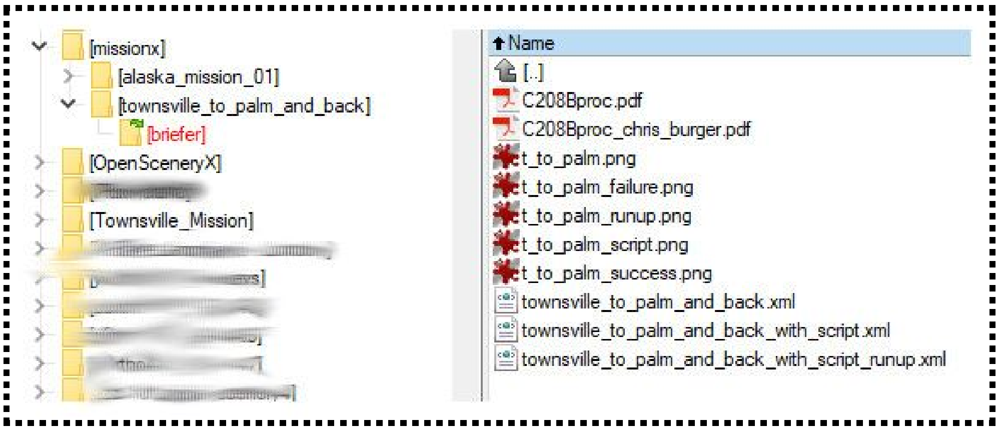
   I suggest to always use a "lowercase letters" rule when naming folders or files, this will solve compatibility differences between OSes.

3. Every mission should have a "briefer" image to distinguish the mission and should have "success" and "failure" images as well.

These core mission files must reside in "{mission folder}/briefer" folder.
The sizes of the expected image files:

- Introductory Image: 290x420 (WxH)
- Success/Failure Images: 975x210 (WxH).
- Map: 1190x490 (WxH)
  The image itself can be a small subset of that space, but that is the maximum expected image size.
  If the image is smaller in size, the plugin will stretch it.

Here is an example to a simple mission folder:



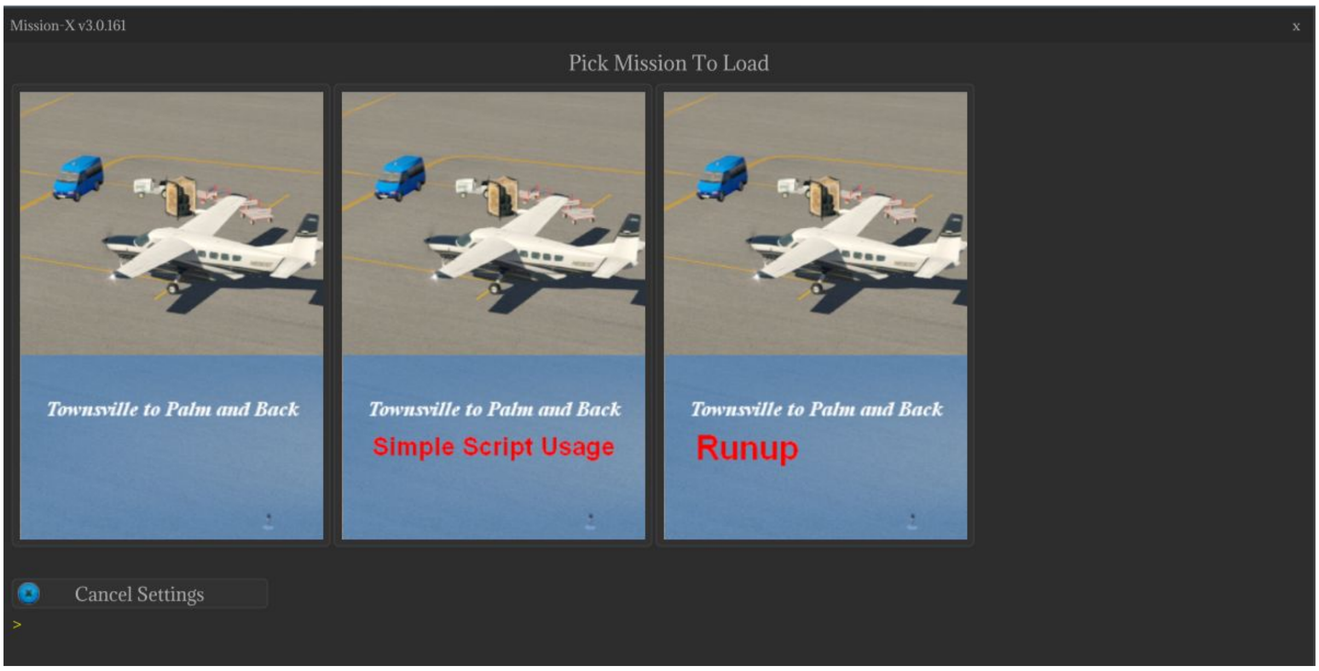
[missionx]
[alaska_mission_01]
[townsville_to_palm_and_back]
[briefer]
[OpenSceneryX]
[Townsville_Mission]

Name
[..]
C208Bproc.pdf
C208Bproc_chris_burger.pdf
t_to_palm.png
t_to_palm_failure.png
t_to_palm_runup.png
t_to_palm_script.png
t_to_palm_success.png
townsville_to_palm_and_back.xml
townsville_to_palm_and_back_with_script.xml
townsville_to_palm_and_back_with_script_runup.xml

If you read the designer guide, you should be aware that you could define other folders for specific file types, but that is only for convenience.
In this example "townsville_to_palm_and_back" is the name of the mission folder.
The files:

- "t_to_palm.png" is a briefer mission intro image **(mission 1)**.
- "t_to_palm_script.png" is a briefer mission intro image **(mission 2)**.
- "t_to_palm_runup.png" is a briefer mission intro image **(mission 3)**.
- "t_to_palm_success.png" is the success image.
- "t_to_palm_failure.png" is the failure image.

Here is an example how it should look in the "Mission List" screen:


Mission-X v3.0.161
x
Pick Mission To Load
Townsville to Palm and Back
Townsville to Palm and Back
Simple Script Usage
Townsville to Palm and Back
Runup
Cancel Settings

The example mission I'm using is based on a converted mission I created for Mission-X v2.

Prepare a Flight Plan

Before starting and writing a mission, you should:

1. Think on the flight plan (high level).
    a. Where you want to start and where to end.
    b. Are there any stops on the way ?
    c. Does the simmer need to do some action during the flight leg ?
2. Think how to break the Flight Plan to objectives and tasks (lower level).
3. Define your mandatory tasks.
4. Understand plugin limitations.

In this tutorial, our flight plan is to:

- Deliver goods from “Townsville Int” to “Palm Island” and
- head back to Townsville Intl.

In terms of mission structure, we can convert this into a

- Two flight leg mission:
    - First leg: deliver goods to palm and
    - Second leg: is to fly back.

  or
- One flight leg with 2 distinctive tasks.
    - First task (mandatory) - deliver goods to palm.
    - Second task (mandatory) - fly back.
      Second task is dependent on the first task.

The hands-on example is a simple mission yet has all we need for a short recreational flight.

In this tutorial I’ll construct the mission using the first technique - we will construct two flight legs.
In order to make this mission, we need to break the flight legs into smaller parts: “objectives” and “tasks”.
From task point of view, we should probably create one mandatory task for each objective.
For other conditions or actions we can create other tasks, some will aim to the simmer and others can aim to the plugin. This way we can create a more immersive mission.

We should first focus on the Objectives and Tasks that will make the mission whole.

Objectives, Tasks and implementation methods

I suggest to create a simple table that will describe Objectives and task/s relations.

Flight Plan:

	Flight Leg description	comments
Leg I	Deliver goods to Palm Island	
Leg II	Deliver goods from Palm to Townsville	Last leg + ending

Objective List:

	Objective Name	Mandatory	comments
A	Fly to Palm	yes	Fly to Palm island and deliver the goods
B	Fly back to Townsville	yes	Fly back to Townsville and back to gateway to deliver Palm goods + ending.

Task List:

	Task Name	mandatory	Implementation component	Comments
01	Land at Palm airport	no	trigger: trig_land_in_YPAM	Will trigger a message that will direct the simmer to reach the terminal area to deliver goods
02	Reach apron area in Palm	yes	trigger: trig_park_in_YPAM	This will trigger a message that will describe goods being delivered. *In v3.0.214 we check speed + parkbreak + inventory (simmer needs to move item in inventory)*
03	Land in Townsville airport	no	trigger: trig_land_in_YBTL	Same as task 01, simple message to direct simmer to the parking location. *[optional]In the future we should place a 3D mark above the gateway so simmer will know exactly where to park.*
04	Reach gateway and deliver Palm goods	yes	trigger: trig_park_in_YBTL	Ending task. *In v3.0.214 we check speed + parkbreak + inventory content (no messing with inventory though)*

Before continuing further, we need to better understand the Task paradigm.
Check the next page.

What is a Task in Mission-X
The Task is *an actual action* that we expect the simmer or the plugin to conduct.
For the plugin it can be a simple way to "call a one time message".
For simmer, an action related to the plane or the mission.
As designers, we need to decide how to implement the rules and actions of the tasks.

In this hands on, we would like in each task, to figure out if the simmer has reached certain location, we basically asking: "has the plane reached a specific area ?", that way we can progress the mission.
In this tutorial, we are using the "trigger" as the main mechanism to test the rules that will progress the mission and send relevant messages, so simmer will know what to do or expect.

If you haven't read "Missionx-X v3 Designer Guide", now is the time to get familiar with the Task, Objective, Leg and Trigger elements and their attributes.

Flight leg and objectives in this mission seem to be similar but a "leg" function is more like a dispatcher of different elements and actions that allow us to organise the progress and content of the current step in the mission. It can drive messages, maps, link to triggers and objectives, it holds the description of what needs to be done and one or more "objectives" that we need to complete a flight leg. Objectives on the other hand, only define tasks (or actions).
The reason of the hierarchy of "leg -> objective -> task" is to allow the designer to mix and match "objectives" and their tasks in different flight "legs", meaning, in some cases you can define the same objective in two different flight "leg" (although the practicality of that can be argued).
The plugin goal is to allow higher flexibility in design, but the price is in readability (which is, I believe, a small price).

Combine Tasks in "objective"s

Our next task as designers is to link "tasks" to the relevant objective and the objectives to the flight leg at hand:

Tasks 01 + 02 will be part of Objective A.
Tasks 03 + 04 will be part of Objective B.

Objective A will be part of Leg I.
Objective B will be part of Leg II.

We will now translate the design into xml elements (use the designer guide to familiarize yourself with the elements and their attributes)

Translate design into Mission-X XML elements

You can start writing any part, the order is not really important to the plugin.
You will see attributes without values, that is acceptable since it means they are not mandatory. You could just remove them from the file for better readability.

```xml
<objectives>
  <objective name="Land and Park in YPAM" title="" >
    <task name="LandInYPAM" title="Land in YPAM" depends_on_task=""
base_on_trigger="trig_land_in_YPAM" base_on_script="" eval_success_for_n_sec="3"
mandatory="" force_evaluation="no"/>
    <task name="ParkInYPAM" title="Park in YPAM" depends_on_task=""
base_on_trigger="trig_park_in_YPAM" base_on_script="" mandatory="yes"
force_evaluation="yes" />
  </objective>

<objective name="Land and Park in YBTL" title="" >
  <task name="LandInYBTL" title="Land in YBTL" depends_on_task=""
base_on_trigger="trig_land_in_YBTL" base_on_script="" eval_success_for_n_sec="3"
mandatory="" force_evaluation="no"/>
  <task name="ParkInYBTL" title="Park in YBTL" depends_on_task=""
base_on_trigger="trig_park_in_YBTL" base_on_script="" mandatory="yes"
force_evaluation="yes" />
  </objective>

</objectives>
```

```xml
<flight_plan>
  <leg name="DeliverPalm" title="Deliver goods to Palm Island" next_leg="FlyBackToYBTL"
>
    <link_to_objective name="Land and Park in YPAM" />
    <desc><![CDATA[It is another day and another delivery is needed to and from Palm
Island. Fly to PALM Island and deliver the goods. Once finish, fly back.]]></desc>
  </leg>

  <leg name="FlyBackToYBTL" title="Fly back to Townsville int" next_leg="" >
    <link_to_objective name="Land and Park in YBTL" />
    <desc><![CDATA[Plane was loaded with postal and other goods. Once ready, Fly back to
Townsville airport YBTL.
    On arrival, taxi to the starting location and shut the mixture.
    Fly safe.]]></desc>
  </leg>
</flight_plan>
```

Some explanations regarding the XML code on previous page

```
<task name="LandInYPAM" title="Land in YPAM" depends_on_task=""
base_on_trigger="trig_land_in_YPAM" base_on_script="" eval_success_for_n_sec="3"
mandatory="" force_evaluation="no"/>
```

<task> Attributes
name: When we define a task, we have to give it a unique name in its objective scope only.
title: The title can be readable text for the user, but currently we do not use the task title (we might use it in the future).
depends_on_task: not in use in this mission
base_on_trigger: we will monitor simmer actions using a trigger. The trigger name will be **"trig_land_in_YPAM"**
base_on_script: we do not use scripts in this mission.
eval_success_for_n_sec: This attribute allows us to monitor a "condition" or "set of conditions" for a time and only then accept and set the "success" event.
mandatory: is this task an optional - "nice to have" or a "must have".
force_evaluation: in some cases, we always want to test a task since we need its condition to always be fulfilled as part of the objective. In this specific task "LandInYPAM", it should only fire once and sees to exist.

```
<objective name="Land and Park in YPAM" title="" >
 <task .... />
</objective>
```

<objective> Attributes
name: Unique objective name
title: Optional for designer readable name

The objective is a container of actions (tasks), so we just need to give it a unique name

Flight <leg> element:

```
<leg name="DeliverPalm" title="Deliver goods to Palm..." next_leg="FlyBackToYBTL" >
    <link_to_objective name="Land and Park in YPAM" />
    <desc><![CDATA[It is another day and another delivery is needed to and from Palm Island. Fly to PALM Island and deliver the goods. Once finished, fly back.]]></desc>
</leg>
```

<leg> Attributes
name: Unique flight leg name
title: *Optional* for user readable name
next_leg: Optional, which flight leg will continue this one once it is achieved.

The <leg> element use the "link_to_objective" and "link_to_trigger" to define the scope of the flight leg. In this mission we only link to one "objective":

```
<link_to_objective name="Land and Park in YPAM" />
```

This is also the scope of the flight leg. We can add other objectives to have a bigger scope.

It is Trigger time

If you read carefully the instructions so far, you should be aware that there are few missing pieces in this mission and it is the conditioning/logic and messaging.
In our Task definitions we decided to use the "base_on_trigger" attribute, which means that we need to define triggers to test plane location and to move the mission forward.

I strongly suggest to go over the "Trigger" topic in the "Designer Guide" document.

Here is an example for triggers definition:

```
<triggers>
  <!-- trig_land_in_YPAM: will be triggered when plane lands in PALM airfield. -->

 <trigger name="trig_land_in_YPAM" type="poly" enabled="yes" >
   <conditions plane_on_ground="yes" cond_script="" />
   <outcome message_name_when_fired="LandedYPAM" message_name_when_left=""
            script_name_when_fired="" script_name_when_left="" post_script="" />
   <loc_and_elev_data>
     <point lat="-18.7510338" long="146.578293"/>
     <point lat="-18.7591095" long="146.584763"/>
     <point lat="-18.7591953" long="146.584625"/>
     <point lat="-18.7511101" long="146.578003"/>
   </loc_and_elev_data>
 </trigger>
```

```
<!-- trig_park_in_YPAM: will be triggered when plane reaches PALM apron area. -->

  <trigger name="trig_park_in_YPAM" type="rad" enabled="yes" >
    <conditions plane_on_ground="yes" cond_script="" />
    <outcome message_name_when_fired="ParkedInYPAM" message_name_when_left=""
             script_name_when_fired="" script_name_when_left="" post_script="" />
    <loc_and_elev_data>
      <radius length_mt="20"/>
      <point lat="-18.7517052" long="146.579651"/>
    </loc_and_elev_data>
  </trigger>


<!-- trig_land_in_YBTL: will be triggered when plane Land in YBTL. -->

  <trigger name="trig_land_in_YBTL" type="poly" enabled="yes" >
    <conditions plane_on_ground="yes" cond_script="" />
    <outcome message_name_when_fired="LandedYBTL" message_name_when_left=""
             script_name_when_fired="" script_name_when_left="" post_script="" />
    <loc_and_elev_data>
      <point lat="-19.258036" long="146.764552"/>
      <point lat="-19.238331" long="146.773902"/>
      <point lat="-19.238432" long="146.774572"/>
      <point lat="-19.258186" long="146.765085"/>
    </loc_and_elev_data>
  </trigger>


<!-- trig_park_in_YBTL: will be triggered when plane returns to starting gateway and
will end the mission. -->

  <trigger name="trig_park_in_YBTL" type="rad" enabled="yes" >
    <conditions plane_on_ground="yes" cond_script="" />
    <conditions plane_on_ground="yes" cond_script="" />
    <outcome message_name_when_fired="EndMissionMsg" message_name_when_left=""
             script_name_when_fired="" script_name_when_left="" post_script="" />
    <loc_and_elev_data>
      <radius length_mt="20"/>
      <point lat="-19.2536716" long="146.770935"/>
    </loc_and_elev_data>
  </trigger>
</triggers>
```

Some explanations regarding the Trigger XML code:

```
<trigger name="trig_land_in_YPAM" type="poly" enabled="yes" >
    <conditions plane_on_ground="yes" cond_script="" />
    <outcome message_name_when_fired="LandedYPAM" message_name_when_left=""
             script_name_when_fired="" script_name_when_left="" post_script="" />
    <loc_and_elev_data>
      <point lat="-18.7510338" long="146.578293"/>
      <point lat="-18.7591095" long="146.584763"/>
      <point lat="-18.7591953" long="146.584625"/>
      <point lat="-18.7511101" long="146.578003"/>
    </loc_and_elev_data>
  </trigger>
```

Trigger Attributes:

<trigger> Attributes
name: Unique trigger name
type: poly, a physical location. Polygonal area can bound a specific area more precisely.
Enabled:is trigger active ? (default is “yes”)
condition: **plane_on_ground:** One of the build in conditions that trigger can test against. In this case we want the plane to be on the ground. **cond_script:** we do not implement script in this example, but if we would have, we would have modified an internal “*script_conditions_met_b*” property using “fn_set_trigger_property()” function to flag triggers condition script as success (see designer guide, “script properties”).

Outcome (replaced = ~~Event_scripts & message~~)
`<outcome message_name_when_fired="LandedYPAM" message_name_when_left=""`
`         script_name_when_fired="" script_name_when_left="" post_script="" />`
 	
The outcome of the trigger is defined in this element, and in this outcome we only call one of the predefined messages.
The message ”LandedYPAM” will be called once we landed in YPAM but only after the plane has been on the ground for 3 seconds (Task: LandInYPAM defines *eval_success_for_n_sec*="3"). If you want an immediate response, you should leave eval_success_for_n_sec empty.
 	
On the other hand if we would like to call a script we would have used the “*script_name_when_fired*” or “*script_name_when_left*” attributes to maybe call special message or set internal DataRefs of X-Plane.
 	
The “*post_script*” attribute is a special case, since it would have been called all the time even if trigger is disabled. Reason: allow you to enable/disable the trigger or other triggers according to certain rules you decide as a designer.

loc_and_elev_data:

Our trigger is polygonal based therefore we need at least three points to define the area of effect. Since we define the trigger condition `plane_on_ground="yes"` then we know the plugin won't call our trigger if plane is still in the air.

If we would like to make sure trigger will fire only when the plane is airborne, then we could have set `plane_on_ground="no"` **or** define an elevation volume, something like the following:

```
<elevation_volume  elev_min_max_ft="100|4000"  />
```

See "elevation_volume" topic in "designer guide"

For triggers that are based on "radius" the definition is simpler.

In trigger `trig_park_in_YPAM`, we define **<loc_and_elev_data>** as follows:

```
<loc_and_elev_data>
      <radius length_mt="20"/>
      <point lat="-19.2536716" long="146.770935"/>
</loc_and_elev_data>
```

Here we define the radius length in meters and the central point of the trigger area. The plugin will calculate these settings (on the ground, and radius area) and call "`script_name_when_fired`" when plane will fulfill all of them.

<messages>

The last piece of the puzzle is the messaging mechanism. In Mission-X v3 we can define messages in two ways, predefined or dynamic ones through script coding.
In most cases I suggest to create messages as part of the <message_template> element, and if you need a dynamic message, just create one message with the name: "dynamic_msg" that you will modify during flight. This is only a suggestion not a rule.

In our tutorial, we call the messaging mechanism to send some informative communication to the simmer. In Mission-X v3 messaging is a little more flexible than in v2.
In v3 all messages are "channels" and in each channel you have 3 tracks: "text", "comm" and "back".

- **Text**: stands for visual text display (most common)
- **Comm**: stand for communication sound file. This is relevant only if you want to play the message text from your own "mp3/wav" sound file.
- **Back**: stands for background. Music that you would like to play in the background, usually its volume should be lower than the communication track.

** The image was taken from "Power Director" to illustrate the track concept.*

In this tutorial we only implemented "text" tracks, since it uses X-Plane own narrator synthesizer.

```
<message name="LandedYPAM" >
    <mix track_type="text" mute_xplane_narrator="" hide_text=""
         override_seconds_to_display_text="" add_minutes="" timelapsed_to_local_hours=""
          set_day_hours="" post_script="" next_msg="" >
         <![CDATA[You landed in Palm Island Airport. Roll to the GA parking
                  area.]]>
    </mix>
</message>
```

The messaging system can be simple - only text messages, or richer - by adding your own soundtracks or defining next message to call to create dialog like interaction. You can even inject a metar file, change the time or call a script (see designer guide).

It is up to you to synchronize the text displayed and sound together using "`override_seconds_to_display_text`" (display time vs play time),
You can leave this to the plugin, but it might not be precise as "hand adjustment".

We have finished designing the flow of the mission and the interaction with the pilot.
It is time to wrap up the mission file, and prepare the last elements so the mission will be usable by the plugin. This means: "MISSION, "mission_info", "global_settings", "briefer" and "end_mission" elements.

<MISSION>

The MISSION tag is the enclosing element of the mission data file.

The "version" was bumped from "300" to "30**1**" since v3.0.242.2

```
<MISSION version="301" name="Townsvile01" title="There and Back" designer_mode="no" >
```

<mission_info>

One liner element.
This element is important for the "mission pick" and "mission detail description" windows, before starting the mission.

```
<mission_info
mission_image_file_name="t_to_palm.png"
plane_desc="GA plane, Cessna C208B - Grand Caravan or equivalent" estimate_time="~45
Minutes."
difficulty="Easy."
other_comments="Mission should not start in cold startup." weather_settings="Weather is
set by mission."
scenery_settings="OpenScenryX v2.0, Townsville_Mission (should be part of the mission
package)"
/>
```

The setup information for our mission is listed in this element. Their role is self explanatory from their name.
Always make sure you define this element to make mission settings easier for the simmer.

<global_settings> element

```
<global_settings>
  <folders sound_folder_name="sound"  />
  <start_time day_in_year="100" hours="22" min="00" />
  <base_weights_kg pilot="85" passengers="0" storage="5" />
</global_settings>
```

In <global_settings>, we can define:

- Mission starting date and time.
- Alternative folders to store different resource files, like sound or 3D object files.
- Base_weights_kg - designer should define basic expected weights for the mission. If not then defaults would be: 85kg, 0kg, 5kg.

In our tutorial we almost do not use any of the global_settings values. The plugin only read the “start_time” and “folders”, but since our mission is quite basic, the use of the sound folder is not relevant.

<briefer> element

The briefer is the “gateway” to your mission. In the briefer element you describe the background for the mission and the settings that should take place. You also define the starting location using ICAO code and even better “latitude” and “longitude” for exact locations.
Let’s take a look at our demo “briefer” element:

```
<briefer starting_icao="YBTL" starting_leg="DeliverPalm" >
  <location_adjust lat="-19.2536716" long="146.770935" elev_ft="0" heading_psi="13.502"
pause_after_location_adjust="true" starting_speed_kts=""/>
  <![CDATA[Hello Simmer.
Townsville to Palm and back is based on a short YouTube video filmed by the pilot
himself (see below for URL).
Main objective is to transport goods from one location and back.
The flight can be based on IFR or VFR rules (movie based on IFR).

Movie URL: http://youtu.be/dd6MQTJz0D4?hd=1

Enjoy, Snagar!]]>
</briefer>
```

Some explanations to the <briefer> XML code:

<briefer> Attributes
starting_leg: unique **flight leg** name to start
starting_icao: optional starting location, it is better to just use “lat/long” in <location_adjust> element
location_adjust: **Lat + long**: exact starting position. For precise location use decimal location with 6 positions after the decimal sign. **elev_ft:** If 0, plugin will probe for ground elevation, Else will try to place at the height defined. **heading_psi:** plane heading, true north. **pause_after_location_adjust:** After plane positioned does X-Plane start in “pause” mode or not. This is probably useful when starting airborn. **starting_speed_kts:** starting speed in Knots.
<![CDATA[ ]]: Detail mission description.

<end_mission>

Every mission should have a successful or failure description. For that we have the <end_mission> element.

```
<end_mission>
  <success_image file_name="t_to_palm_success.png"  />
  <success_msg ><![CDATA[well done pilot. all goods... ]]></success_msg>
  <success_sound sound_file="" sound_vol="" />

  <fail_image file_name="t_to_palm_failure.png"  />
  <fail_msg ><![CDATA[You failed the mission.]]></fail_msg>
  <fail_sound sound_file="" sound_vol="" />
</end_mission>
```

For every mission ending we have an image, some message and optional sound file.

Some explanations to the <end_mission> XML code

<ending_mission> attributes
"success_image": file name to display when Mission is flagged as "success". "success_msg": a message that will be displayed in the "briefer detail" area. "success_sound": should be played when the briefer displays the message. The same applies to failure elements.

Example for mission data file based on Mission-X v3.0.242.2

```
<?xml version="1.0" encoding="UTF-8"?>
<MISSION name="Townsville_to_Palm" title="Townsville to Palm and back" designer_mode="no"
version="301" >
  <mission_info mission_image_file_name="t_to_palm.png" plane_desc="GA plane, Cessna C208B - Grand
Caravan or equivalent" estimate_time="~45 Minutes." difficulty="Easy."
  other_settings="Mission should not start in cold startup." weather_settings="Weather is set by
mission." scenery_settings="OpenScenryX v2.0, Townsville_Mission (should be part of the mission
package)" />

  <global_settings>
    <folders sound_folder_name="sound" />
    <start_time days="100" hours="22" min="00" />
  </global_settings>


  <!-- Briefer  -->
  <briefer starting_icao="" starting_leg="DeliverPalm" >
    <location_adjust lat="-19.2536716" long="146.770935" elev_ft="0" heading_psi="13.502"
                     pause_after_location_adjust="true" starting_speed_kts=""/>
    <![CDATA[Hello Simmer.
Townsville to Palm and back is based on a short YouTube video filmed by the pilot himself (see below
for URL).
Main objective is to transport goods from one location and back.
The flight can be based on IFR or VFR rules (movie based on IFR).

Movie URL: http://youtube/dd6MQTJz0D4?hd=1

Enjoy, Snagar!]]>
  </briefer>

  <flight_plan>
    <leg name="DeliverPalm" title="Deliver goods to Palm Island" next_leg="FlyBackToYBTL" >
      <link_to_objective name="Land and Park in YPAM" />
      <desc><![CDATA[It is another day and another delivery is needed to and from Palm Island.
      Fly to PALM Island and deliver the goods. Once finish, fly back.]]></desc>
    </leg>

    <leg name="FlyBackToYBTL" title="Fly back to Townsville int" next_leg="" >
      <link_to_objective name="Land and Park in YBTL" />
      <desc>
      <![CDATA[After the plane was loaded with postal and other goods, you set your plane and
       head to the runway.
       Fly back to Townsville airport YBTL.

       On arrival, taxi to the starting location and shut the mixture.
       Fly safe.]]>
      </desc>
    </leg>

  </flight_plan>




<objectives>
```

```
      <objective name="Land and Park in YPAM" title="" >
        <task name="LandInYPAM" title="Land in YPAM" depends_on_task=""
base_on_trigger="trig_land_in_YPAM" base_on_script="" eval_success_for_n_sec="3" mandatory=""
force_evaluation="no"/>
        <task name="ParkInYPAM" title="Park in YPAM" depends_on_task=""
base_on_trigger="trig_park_in_YPAM" base_on_script="" mandatory="yes" force_evaluation="yes" />
        <desc><![CDATA[Park plane in YPAM, and deliver the goods.]]></desc>
      </objective>

    <objective name="Land and Park in YBTL" title="" >
      <task name="LandInYBTL" title="Land in YBTL" depends_on_task=""
base_on_trigger="trig_land_in_YBTL" base_on_script="" eval_success_for_n_sec="3" mandatory=""
force_evaluation="no"/>
      <task name="ParkInYBTL" title="Park in YBTL" depends_on_task=""
base_on_trigger="trig_park_in_YBTL" base_on_script="" mandatory="yes" force_evaluation="yes" />
      <desc><![CDATA[Fly back to Townsville int. Park plane in YBTL to finish.]]></desc>
    </objective>

   </objectives>

  <triggers>
    <trigger name="trig_land_in_YPAM" type="poly" enabled="yes" >
      <conditions plane_on_ground="yes" cond_script="" />
     <outcome message_name_when_fired="LandedYPAM" message_name_when_left=""
              script_name_when_fired="" script_name_when_left="" post_script="" />
      <loc_and_elev_data>
        <point lat="-18.7510338" long="146.578293"/>
        <point lat="-18.7591095" long="146.584763"/>
        <point lat="-18.7591953" long="146.584625"/>
        <point lat="-18.7511101" long="146.578003"/>
      </loc_and_elev_data>
    </trigger>

    <trigger name="trig_park_in_YPAM" type="rad" enabled="yes" >
      <conditions plane_on_ground="yes" cond_script="" />
     <outcome message_name_when_fired="ParkedInYPAM" message_name_when_left=""
              script_name_when_fired="" script_name_when_left="" post_script="" />
      <loc_and_elev_data>
        <radius length_mt="20"/>
        <point lat="-18.7517052" long="146.579651"/>
      </loc_and_elev_data>
    </trigger>

    <trigger name="trig_land_in_YBTL" type="poly" enabled="yes" >
      <conditions plane_on_ground="yes" cond_script="" />
     <outcome message_name_when_fired="LandedYBTL" message_name_when_left=""
              script_name_when_fired="" script_name_when_left="" post_script="" />
      <loc_and_elev_data>
        <point lat="-19.258036" long="146.764552"/>
        <point lat="-19.238331" long="146.773902"/>
        <point lat="-19.238432" long="146.774572"/>
        <point lat="-19.258186" long="146.765085"/>
      </loc_and_elev_data>
    </trigger>

    <trigger name="trig_park_in_YBTL" type="rad" enabled="yes" >
      <conditions plane_on_ground="yes" cond_script="" />
     <outcome message_name_when_fired="EndMissionMsg" message_name_when_left=""
              script_name_when_fired="" script_name_when_left="" post_script="" />
      <loc_and_elev_data>
        <radius length_mt="20"/>
        <point lat="-19.2536716" long="146.770935"/>
      </loc_and_elev_data>
    </trigger>
  </triggers>


  <message_templates>

    <message name="LandedYPAM" >
        <mix track_type="text" mute_xplane_narrator="" hide_text=""
override_seconds_to_display_text="" enabled="" ><![CDATA[You landed in Palm Island Airport. Roll to
the GA parking area.]]></mix>
    </message>

    <message name="ParkedInYPAM" >
        <mix track_type="text" mute_xplane_narrator="" hide_text=""
override_seconds_to_display_text="30" enabled="" ><![CDATA[You reached the YPAM delivery area. Stop
plane and deliver the cargo. When you are ready, head back to Townsville Int.]]></mix>
    </message>

    <message name="LandedYBTL" >
        <mix track_type="text" mute_xplane_narrator="" hide_text=""
override_seconds_to_display_text="" enabled="" ><![CDATA[You landed in Townsville Airport. Roll to
the "missionx ramp" in the GA parking area and stop plane.]]></mix>
    </message>

    <message name="EndMissionMsg" >
        <mix track_type="text" mute_xplane_narrator="" hide_text=""
override_seconds_to_display_text="30" enabled="" ><![CDATA[Well done. You parked the plane and
delivered the goods.]]></mix>
    </message>

  </message_templates>

  <end_mission>
    <success_image file_name="t_to_palm_success.png"  />
    <success_msg ><![CDATA[well done pilot. all goods have been handed to the relevant people. Have a
nice day and see you tomorrow. ]]></success_msg>
    <success_sound sound_file="" sound_vol="" />

    <fail_image file_name="t_to_palm_failure.png"  />
    <fail_msg ><![CDATA[You failed the mission.]]></fail_msg>
    <fail_sound sound_file="" sound_vol="" />
  </end_mission>


</MISSION>
```

Revision Table

Revision No.	Date + Modifier	What was changes
0.07	18-feb-2020	Fixed mission version attribute to be “301” as of v3.0.242.2
0.06	29-aug-2019	Made the document compatible with the latest element/attributes name and workflow.
0.05	23-nov-2018	Replaced: `message_name_when_enter` and `script_name_when_enter` with “`message_name_when_fired`” and “`script_name_when_fired` ” (v3.0.214). Minor explanation rewrites.
0.04	06-jul-2018	Replaced “always_evaluate” with “force_evaluation” (v3.0.206)
0.03	13-jan-2018	Some fixes and adjustments to v3.0.19x
0.02	25-dec-2017	First publishing